



The Transport Layer

Redes de Computadores

2021/22

Pedro Brandão

1

References

- These slides are from “Computer Networking: A Top Down Approach 5th edition. Jim Kurose, Keith Ross Addison-Wesley, April 2009”
 - With adaptations/additions by Manuel Ricardo and Pedro Brandão

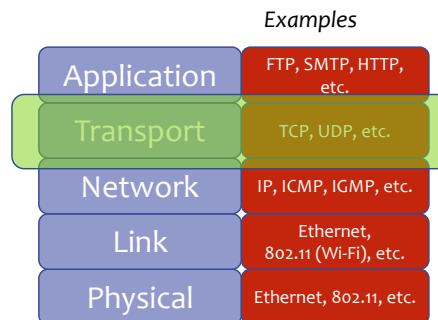
Driving questions...

- What are the services provided by the Transport Layer?
- What are the transport protocols in the TCP/IP stack?
- What are the differences between UDP and TCP?
- How is the connection established in TCP?
- What is the difference between flow control and congestion control?
- How does TCP implement flow control?
- What mechanisms does TCP adopt to prevent network congestion control?
- Why is it the congestion control mechanism implemented by TCP so important for the behaviour of the Internet?

3

Internet protocol stack

- Application: network processes
- Transport: data transfer between processes
- Network: packet routing between source and destination
- Link: data transfer between adjacent network elements
- Physical: bits on the “wire”



4

Processes communicating

Process: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Comunicações Bidimensionais → permitem a comunicação de um lado para outro e cada um dos extremos executa a função que lhe foi atribuída na aplicação que corre em cada um dos lados

localmente

Quem ativamente inicia a comunicação

Client process: process that actively initiates communication

Server process: process that passively waits to be contacted

→ Quem passivamente espera para ser comunicado. já tem uma porta aberta
e está à escuta.

troca de informação

5

Transport vs. network layer

- network layer: logical communication between hosts
- transport layer: logical communication between processes
 - relies on, enhances, network layer services

Analogy:

Mailing between companies

- processes = workers
- app messages = letters in envelopes
- hosts = buildings
- network-layer protocol = postal (CTT) service
- transport protocol = internal office mail distribution worker

Quem vai fazer a entrega aos processos dentro da mesma máquina

6

Addressing processes

- to receive messages, process must have identifier
- host device has unique 32-bit IP address → IP identifica a máquina
- Q: does host IP address on which process runs suffice for identifying the process?
 - A: No, many processes can be running on same host

→ IP não é suficiente

- Identifier includes both IP address and port numbers associated with process on host.

Permite identificar qual o processo que vai estar à escuta da mensagem

→ Deverá receber a mensagem

- Example port numbers:

- ssh server: 22
- SMTP server: 25
 - With TLS: 465

Isso implica que temos um serviço ssh à escuta, nós podemos criar um processo e metê-lo à escuta na porta 22.

- To connect to web server

www.up.pt:

- IP address: 104.18.6.105
- Port: 443
 - For https, it would be 80 for http

Portas reservadas: Abaixo de 1024 tem de ter permissões de root

P

7

Sockets — estrutura que permite, no ponto de vista das máquinas, na camada de transporte, definir que um processo está à escuta nessa socket

Teremos de enviar algo que identifique qual é o processo de destino (em quem estamos a comunicar)

Multiplexing/demultiplexing

Demultiplexing at rcv host:

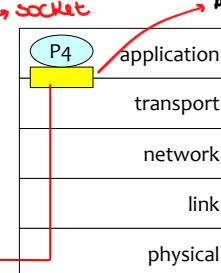
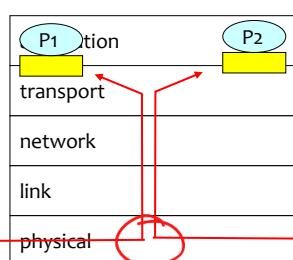
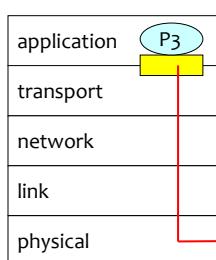
delivering received segments to correct socket

= socket

= process

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



A porta tem de ser claramente identificada

host 2

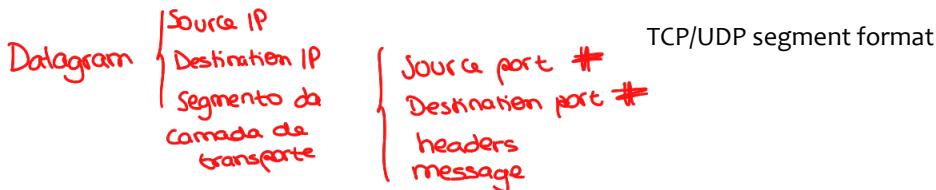
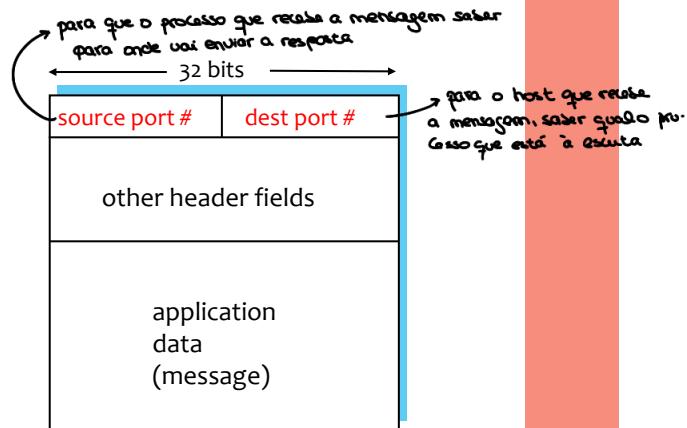
Toda de fazer a distinção para quem são as mensagens, baseando-se nas portas em que P1 e P2 estão à escuta.

Processos em diferentes máquinas a tentar comunicar

8

How demultiplexing works

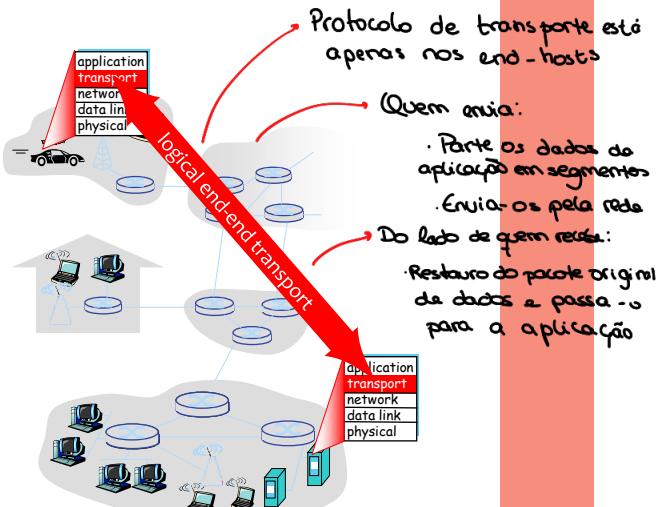
- host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries 1 transport-layer segment
 - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



9

Transport services and protocols

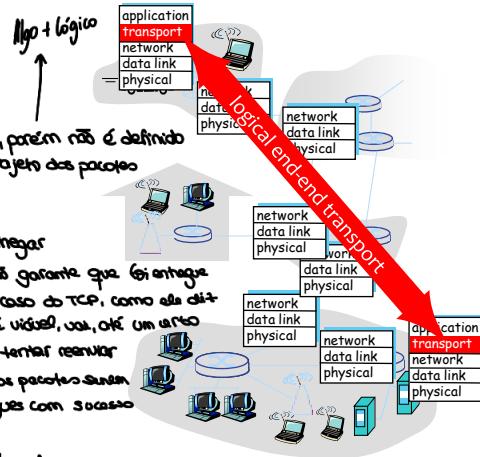
- provide logical communication between app processes running on different hosts
- transport protocols run in end systems
 - send side: breaks app messages into segments, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
 - Internet: TCP and UDP



10

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup



- unreliable, unordered delivery:

UDP → Não é fiável, pois os pacotes podem não chegar
Algo + simples → Dá uma extensão "best-effort" ao IP →

- services not available:
 - delay guarantees
 - bandwidth guarantees

Não há nada que seja provisoriado, no ponto de vista da Comunicação de transporte, para este 2 fatores de qualidade e controlo

11

Internet transport protocols services

TCP service:

- connection-oriented: setup required between client and server processes
- reliable transport between sending and receiving process
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum throughput guarantees, security

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security
- Q: why bother? Why is there a UDP?

Há serviços/aplicações que não vão precisar desta fiabilidade.
Nesse caso podem utilizar algo + simples onde não haja overhead nos cabeçalhos.

P

12

UDP

RFC 768 User Datagram Protocol

(Transport layer)

13

13

UDP - User Datagram Protocol (UDP)

- Allows applications
 - to interface directly to IP
 - with minimal additional protocol overhead
- often used for streaming multimedia apps
 - loss tolerant → Ao perder alguns dos pacotes, apenas se perde um pouco de Qualidade.
 - rate sensitive → No ponto de vista do utilizador ele continua a ouvir algo, põem a qualidade para baixar um pouco (continua a ouvir)
- other UDP uses
 - DNS
 - SNMP
- reliable transfer over UDP: add reliability at application layer
 - application-specific error recovery!

Algo + básico no ponto de vista do acesso à camada de rede

Permite a identificação do processo que tem de receber e do processo que está a enviar os dados

Mais importante que os pacotes vão chegando à altura certa

São muito sensíveis ao débito

A aplicação é que trata de garantir fiabilidade se assim o desejam

Protocolos de transporte deste tipo de frames têm em conta que há frames + importantes que outros no sentido de definir a imagem e os movimentos.

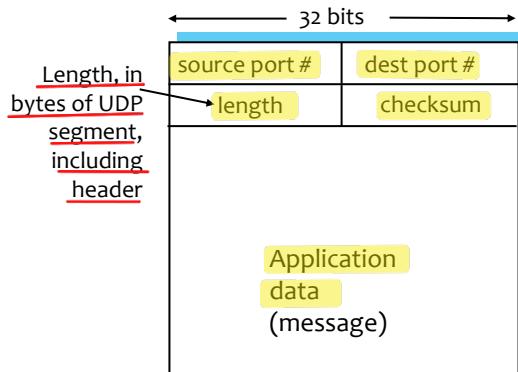
14

14

UDP segment format

- **UDP header**

- Port numbers identify sending and receiving processes
- Checksum covers header and data; optional



15

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

→ Não há limites do ponto de vista do congestionamento. No momento do envio não se preocupa com prever se o envio dos dados vai congestionar o serviço.

llegando ao outro lado, o próximo é algo totalmente independente
Vai enviando datagramas que serão transportados pelo IP
Ao enviar um pacote, o próximo, não está, do ponto de vista da camada UDP, relacionado com o anterior

Do ponto de vista do transporte só os datagramas.
Não há qualquer estado associado ao estabelecimento da conexão

16

TCP

RFC 793 Transmission Control Protocol

(Transport layer)

17

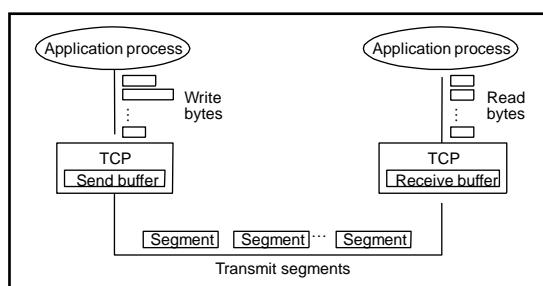
No caso do UDP, não alguém que envia de um lado para o outro e o segmento seguinte não tem nada a ver com o anterior. Se quisesse ter uma comunicação bidirecional, o lado de quem recebeu, teria de enviar para uma porta também aberta do lado de quem enviou. Tente de abrir 2 portas para a ligação.

TCP – Transmission Control Protocol

- **full duplex data:** conexão estabelecida → bi-direcional, independentemente da quem inicia →
 - bi-directional data flow in same connection
- **connection-oriented:**
 - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- **flow controlled:**
 - sender will not overwhelm receiver
 - ARQ mechanism
- **Congestion control**
 - Avoids network's congestion
- **point-to-point:** → De um processo para outro processo
 - one sender, one receiver

- No caso do UDP é ao datagram
- **reliable, in-order byte stream:**
 - no "message boundaries"
 - **pipelined:**
 - TCP congestion and flow control set window size
 - **send & receive buffers**

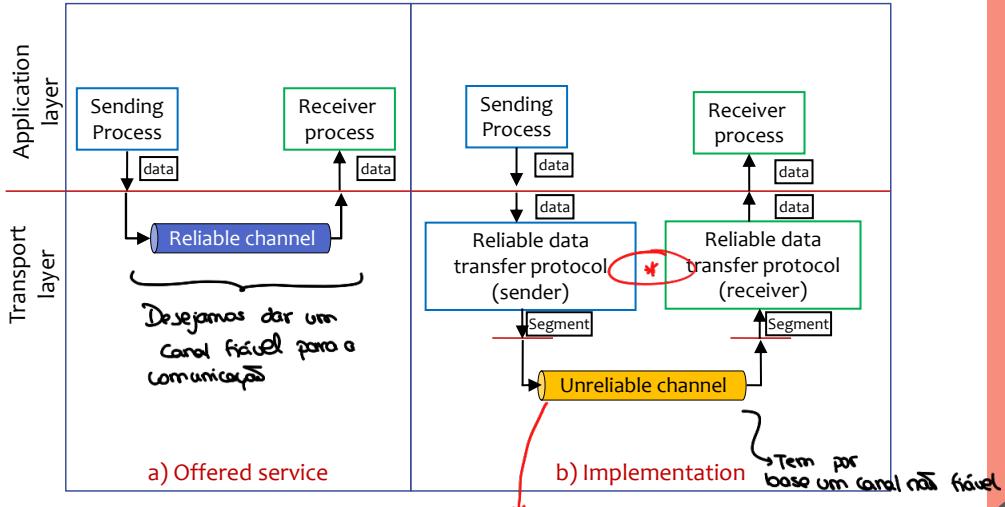
enviamos dados que não são delimitados do ponto de vista da comunicação de transporte, mas só dados que vão estar a ser enviados para o outro lado. Não há fronteiras associadas a este stream de dados



18

18

Principles of reliable data transfer



* Vai ter de existir uma abstração entre o receptor e o emissor sobre o que é a manutenção da fiabilidade e da entrega de todos os pacotes que estão submetidos

19

Reliable transfer: problems and solutions

Packets may be lost

- Receiver confirms (ACK) reception of each segment *Confirmação da recepção para cada um dos segmentos enviados*
- If sender does not receive confirmation within a time limit → re-sends segment

Receiving duplicated segments:

- generated by the network or re-transmissions by lost ACKs
- Segments are numbered
- ACK identifies the number of the last received segment *Na ordem completa — envia ACK se o segmento é o esperado em relação ao Recebido anteriormente*
- It also solves the re-ordering problem

Peenquiamos o ACK

Packets may contain errors

- Detected using checksum
- Packets with errors are ignored → handled as lost packets

Não há confirmação

Pretensão das dados ← *Timeout do lado do sender*

20

Basic TCP Operation

- **Sender**

- Application data is broken in segments
- TCP uses timer while waiting for an ACK of every segment sent
- Un-ACKed segments are **retransmitted**

Retransmite → *Não recebeu*

Parte pedidos da aplicação em segmentos

Há um limite associado ao segmento

Este máximo é o máximo que pode ser transportado pelo rede nas camadas abaixo

→ Espera a confirmação

- **Receiver**

- Errors detected using a checksum
- Correctly received data is acknowledged
- Segments reassembled in proper order
- Duplicated segments discarded

Detectar de erros → Erros são ignorados

Se foram recebidos envia ACK

Duplicados

Confirma com ACK

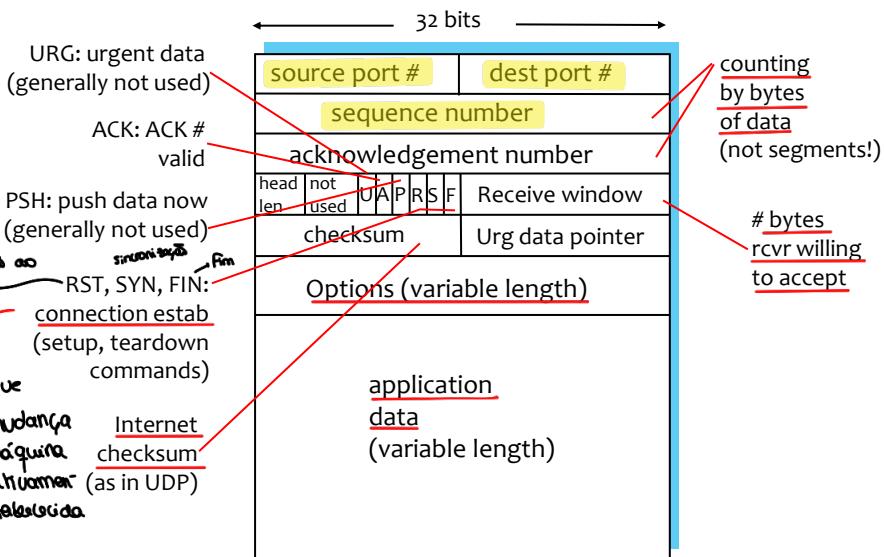
Construção do que eram os dados

Envia para a aplicação

- **Window based flow control**

21

TCP segment structure



22

TCP Header

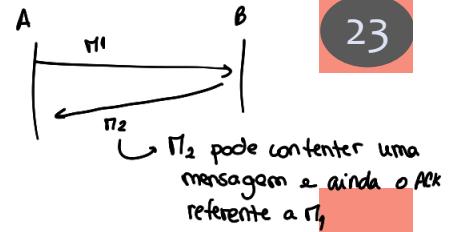
- Ports number are the same as for UDP
- 32 bit SeqNumber uniquely identifies the application data contained in the TCP segment
 - SeqNumber is in bytes
 - It identifies the first byte of data

Identifica qual o 1º byte novo que estamos a transmitir
- 32 bit AckNumber is used for piggybacking ACKs
 - AckNumber indicates the next byte the receiver is expecting
 - Implicit ACK for all of the bytes up to that point

Indica o próximo byte à espera
Se receber até ao 3500, no ACK vai o 3501
- Window size
 - Used for flow control (ARQ) and congestion control
Sender cannot have more than a window of bytes in the network
 - Specified in bytes

Dados pendentes → Sua confirmação
Emissor não pode conter pendentes na rede, mais dados do que o tamanho desta janela

Pode aproveitar uma mensagem que envia novos dados para confirmar uma mensagem recebida:
- Checksum covers the header and data



23

Sequence Numbers in TCP

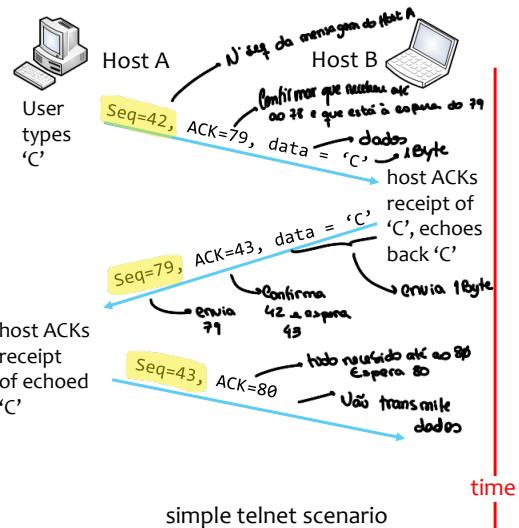
- TCP regards data as a byte-stream
 - each byte in stream is numbered sequentially
- TCP breaks byte stream into segments
 - size limited by the Maximum Segment Size (MSS)
- Each packet has a sequence number
 - sequence number of the 1st byte of data transported by the segment
- TCP connection is duplex
 - data in each direction has different sequence numbers

Como a comunicação é bidirecional: Num sentido temos um número de seqüência, no outro sentido temos outro

24

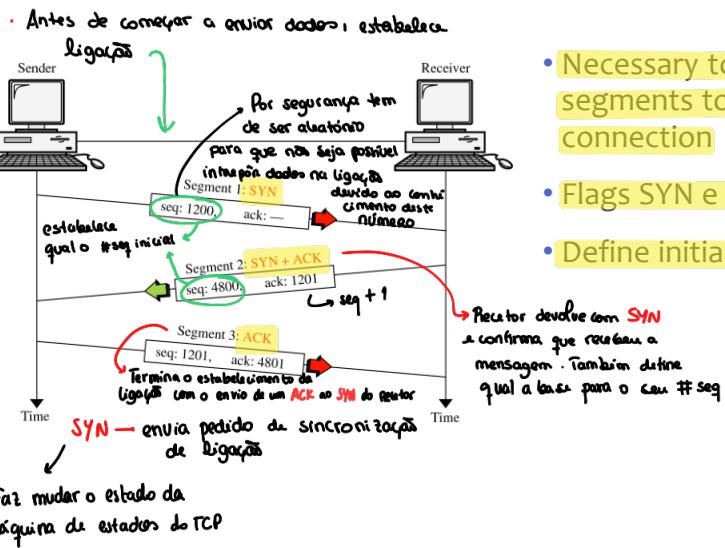
TCP seq. #'s and ACKs

- Seq. #'s:
 - byte stream “number” of first byte in segment's data
- ACKs:
 - seq # of next byte expected from other side
 - cumulative ACK



25

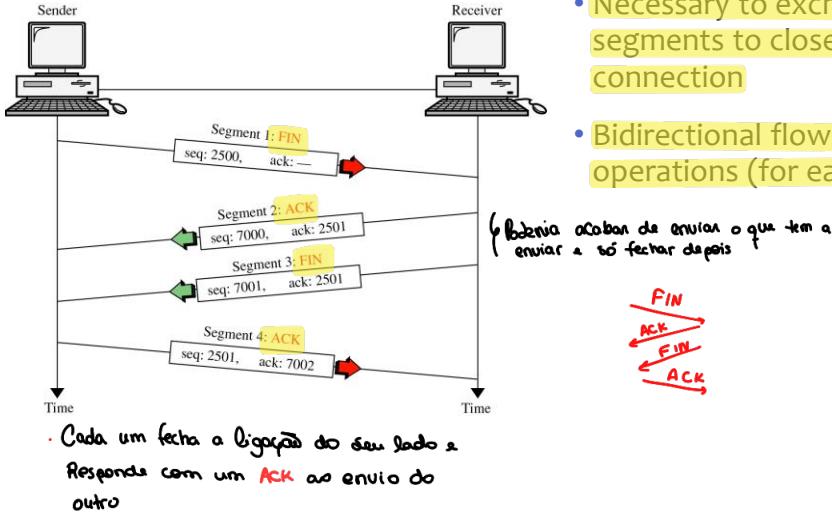
TCP: Connection establishment



- Necessary to exchange 3 segments to establish a TCP connection
- Flags SYN e SYN+ACK
- Define initial sequence numbers

26

TCP: Connection close



- Necessary to exchange 4 segments to close a TCP connection
- Bidirectional flow → 2 closing operations (for each direction)

27

27

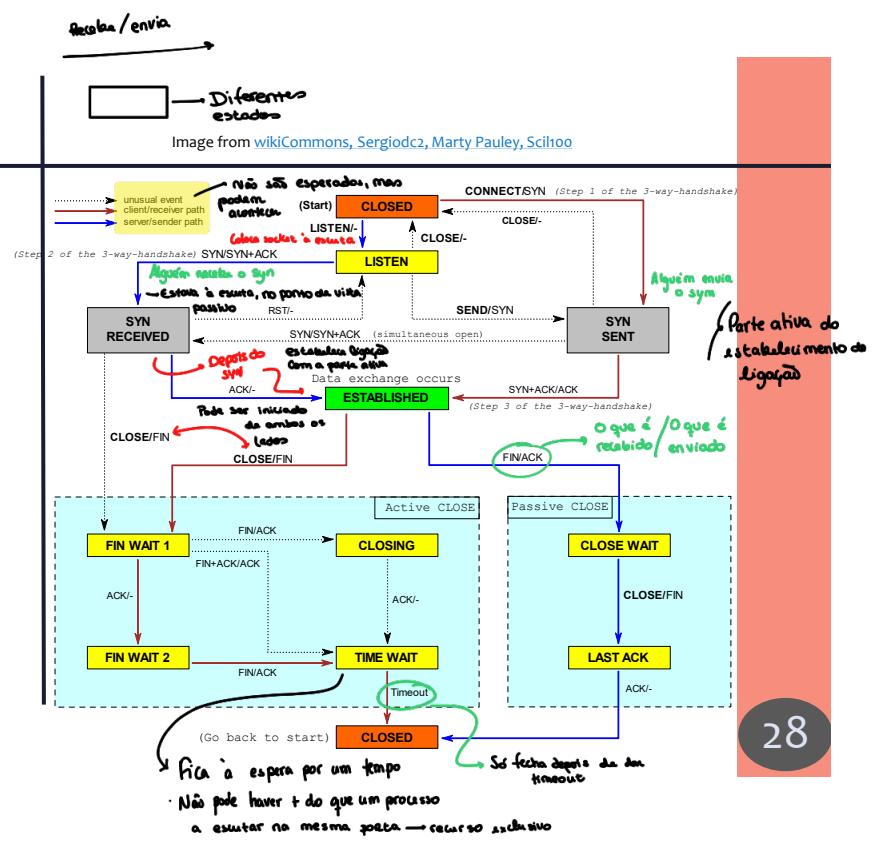
TCP Connection Management FSM

The heavy solid line is the normal path for a client.

The heavy dashed line is the normal path for a server.

The light lines are unusual events.

Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.



28

28

Retransmissions in TCP – A variation of Go-Back-N

- **Sliding window**

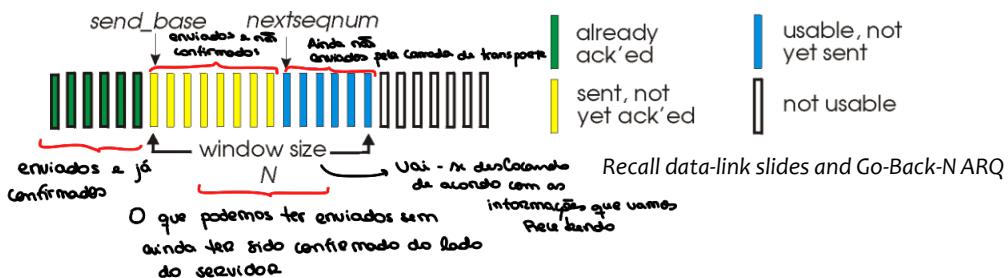
- Ack contains a single sequence number
- acknowledges all bytes with a lower sequence number
- duplicate ACKs sent when out-of-order packet received

Confirma todos os bytes até ao enviado → o que estamos à espera
Se foram enviados pacotes fora da ordem esperada o ACK

- **Sender retransmits a single packet at a time**

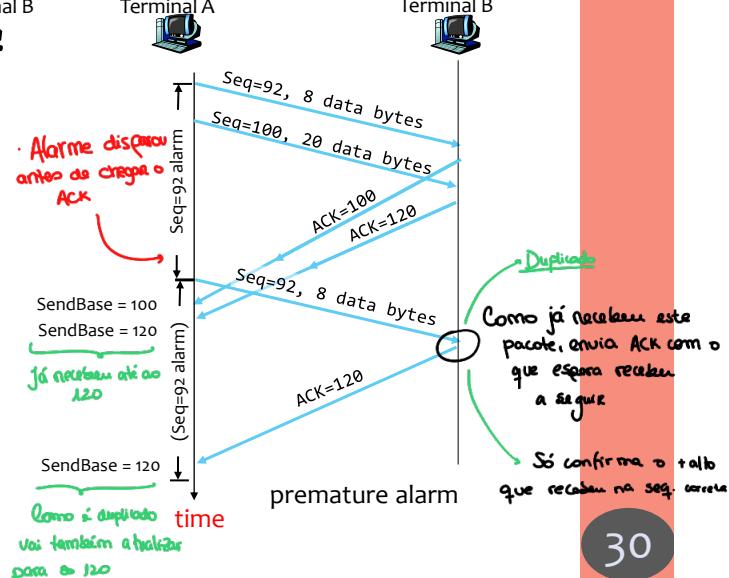
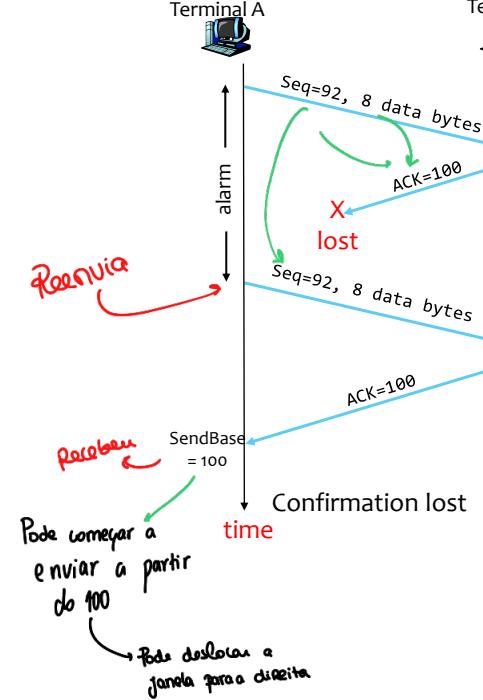
- optimistic assumption → only one packet is lost \rightarrow só se perde 1

- **Error control based on byte sequences, not packets**



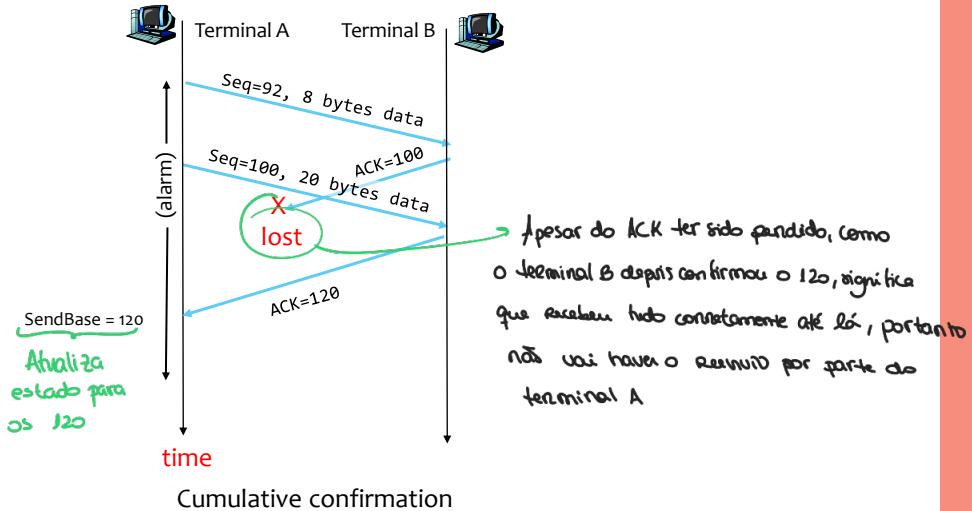
29

TCP: retransmission scenario



30

TCP: retransmission scenarios



31

Adaptive Retransmission

- RFC 6298 - Computing TCP's Retransmission Timer
- RTT → Round Trip Time — tempo que demoramos a "ir e vir"
- sampleRTT measured for each segment/ACK pair
- Moving average (smooth RTT) of SRTT
 - $SRTT_{new} = (1 - \alpha) \times SRTT_{old} + \alpha \times \text{Sample RTT}$
 - $\alpha = 1/8$ 2 passos para determinar os que têm influência.
- Variation of RTT → o que é que ela varia de um para o outro no cálculo do RTT → utilizam isso na estimativa do RTT
 - $RTT_{new} = (1 - b) * RTT_{old} + b * |SRTT_{old} - \text{Smooth RTT}|$
 - $b = 1\%$ para definir as influências que deve ter no cálculo
- $RTT = SRTT + \max(G, K * RTT_{new})$
 - clock granularity of G seconds
 - $K=4$
 - $RTT = \text{retransmission timeout}$

$$RTT = \text{Moving average} + \text{Potencial Variação}$$

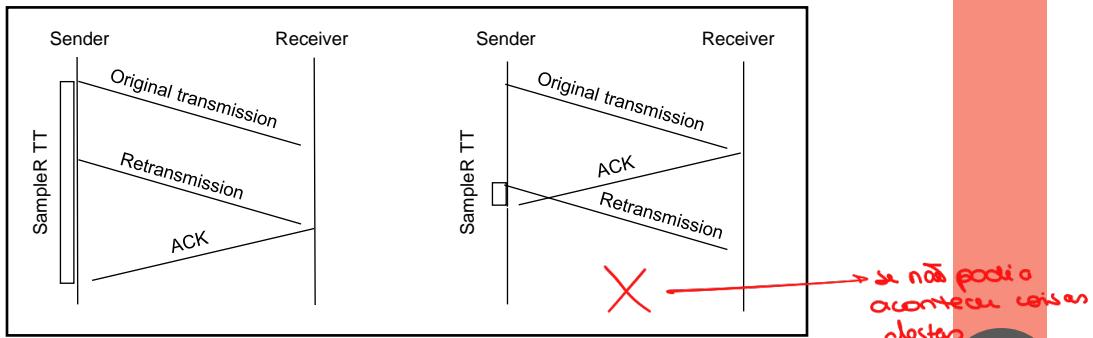
32

Todo de ir tentando estimar o que é o nosso RTT → tempo de retransmissão dos segmentos → é calculado baseando-se no tempo que demora a ir e vir (RTT)

32

Karn/Partridge Algorithm

- sampleRTT not measured in retransmission
- Timeout doubled for each retransmission



→ No cálculo de ocorrer uma retransmissão o valor utilizado para a estimativa é o valor desde que se enviou o primeiro segmento até receber o ACK correspondente.

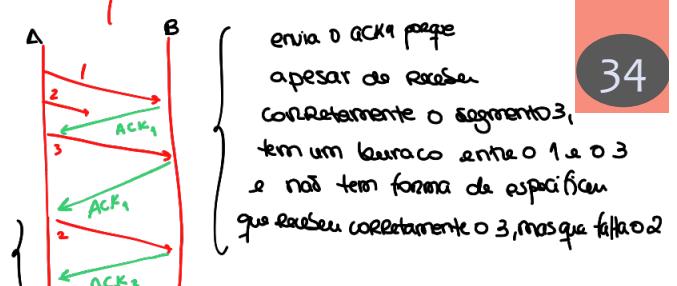
33

Selective ACK

- Option for selective ACKS (SACK) also widely deployed
- Selective acknowledgement (SACK)
 - adds a bitmask of packets received
 - implemented as a TCP option
- When to retransmit?
 - packets may experience different delays
 - still need to deal with reordering
 - wait for out of order by 3 packets

• Dá a possibilidade de especificar quais dos últimos segmentos que receberam condições para não ter de enviar um ACK a dizer que receberam todos.

→ Isto acontece se o receptor B especificasse quais os segmentos que receberam corretamente e quais que estão em falta.

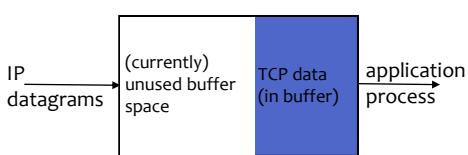


Podia acontecer isto ou podia acontecer de o A voltar a enviar o segmento 3 também se o B tiver de ignorar.

34

TCP Flow Control

- receive side of TCP connection has a receive buffer:



- app process may be slow at reading from buffer

Aplicações leem buffer à medida que lê os dados

A stack protocolar tem que ir mantendo os dados até à aplicação ler-nos podendo deserta-las até lá

- Se a stack tiver o receiver + dados do que aquela que a aplicação lê, irá ocorrer um over-flow do buffer

Adaptação da velocidade de envio do emissor à velocidade de leitura do Receptor

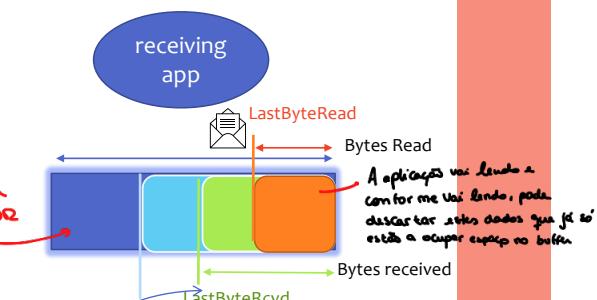
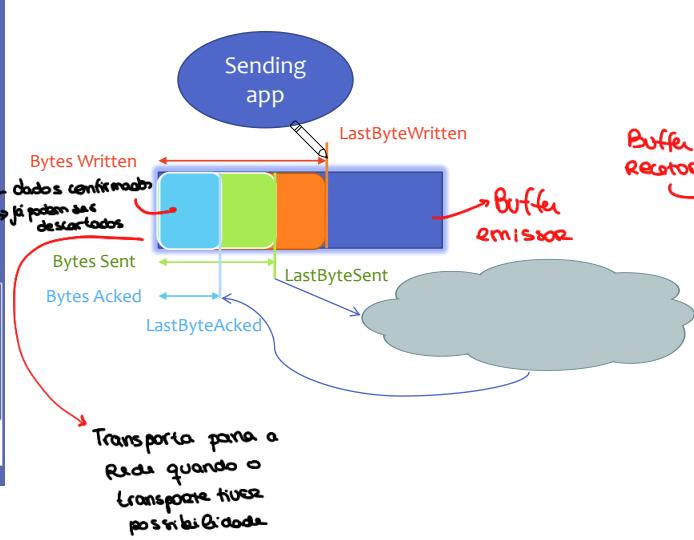
- speed-matching service: matching send rate to receiving application's drain rate
- Receiver informs of buffer free space → Receiver window do TCP header
- Sender limits data in transit to that window

flow control

sender won't overflow receiver's buffer by transmitting too much, too fast

35

TCP Buffers and sliding window



36

Sliding Window

- Sender

- $\text{LastByteAcked} \leq \text{LastByteSent}$ → não se pode enviar um ACK de algo que ainda não se enviou
- $\text{LastByteSent} \leq \text{LastByteWritten}$ → não se pode enviar algo que ainda não foi escrito para o buffer
- Buffers bytes between LastByteAcked and LastByteWritten

Vai descartando os bytes contínuos

- Receiver

- $\text{LastByteRead} < \text{NextByteExpected}$
- $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
- Buffers bytes between LastByteRead e LastByteRcvd

Vai descartando os bytes lidos

37

Flow Control

- Buffer length → 2 limites

- Sender → MaxSendBuffer
- Receiver → MaxRcvBuffer

Tudo aquilo que ainda tem por len

$\text{LastByteRcvd} - \text{LastByteRead}$

< MaxLength

- Receiver

- $\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$
- $\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})$

O que vai anunciar para o espaço livre do buffer é o MaxLength - o que ainda não foi lido escrito confirmado

- Sender

- $\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$

$\text{LastByteWritten} - \text{LastByteAcked}$

- $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$

$\text{LastByteSent} - \text{LastByteAcked}$

- $\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$

Tudo aquilo que se encontra no Buffer do Receptor

O que ainda pode enviar

- Sending application blocks if it needs to write y bytes and

- $(\text{LastByteWritten} - \text{LastByteAcked}) + y \leq \text{MaxSenderBuffer}$

O que não pode diminuir do buffer novos Bytes

Bloqueia se tem + dados para escrever e não tem + espaço

< AdvertisedWindow <

38

Máximo que pode enviar: $\text{MaxSenderBuffer} - (\text{LastByteWritten} - \text{LastByteAcked})$

Fast Retransmit

lata quando há um RTO (calculado anteriormente) muito longo

RTO

- time-out period often relatively long:
 - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
 - sender often sends many segments back-to-back
 - if segment is lost, there will likely be many duplicate ACKs for that segment

em vez de esperar pelo timeout

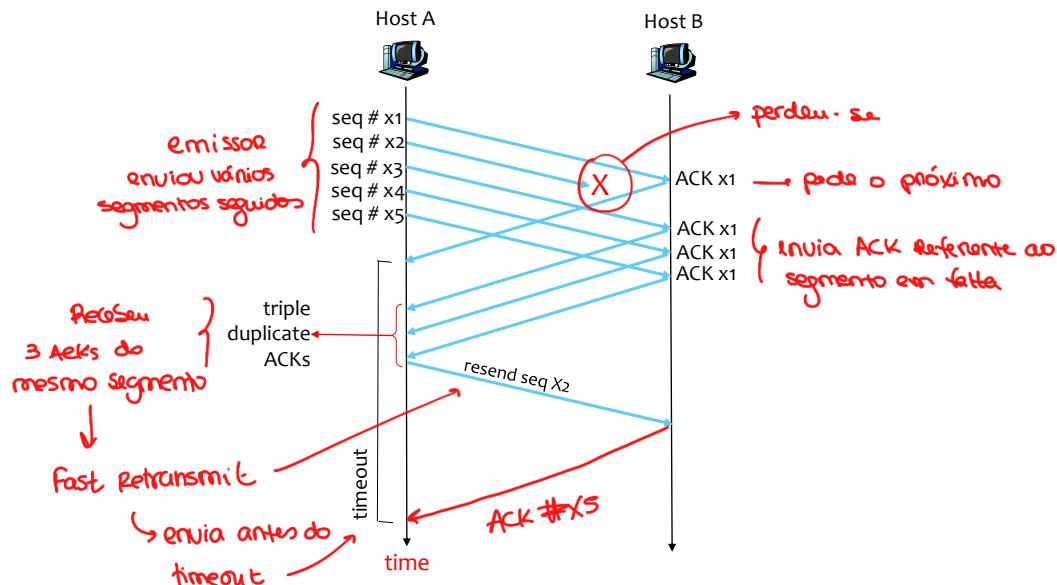
- If sender receives 3 ACKs for same data, it assumes that segment after ACKed data was lost:

- fast retransmit: resend segment before timer expires

emissor eniou muitos segmentos juntos e ainda não tem confirmação
se um deles for perdido vai haver muitos ACKs referentes a esse segmento, pois todos os segmentos à frente desse que foram todos, vão enviar um ACK referente a esse pacote, pois têm no caminho

39

Fast retransmit example



40

Principles of Congestion Control

Congestion: → Há vários emissores a enviar dados e estão a congestionar a rede
→ Há mais utilização da rede do que aquilo que a rede oferece

- informally: “too many sources sending too much data too fast for network to handle”

- different from flow control!

- manifestations:

- lost packets (buffer overflow at routers)
- long delays (queuing in router buffers)

Buffer do Router pode ser suficientemente grande, mas há tantos dados para transmitir → há um atraso na utilização de uma porta por haver tantos dados para transmitir

o emissor está a enviar mais depressa do que aquilo que o receptor consegue consumir

→ se já tiver o buffer cheio não de descontar os novos pacotes recebidos → Há a perda desses pacotes



41

Congestion

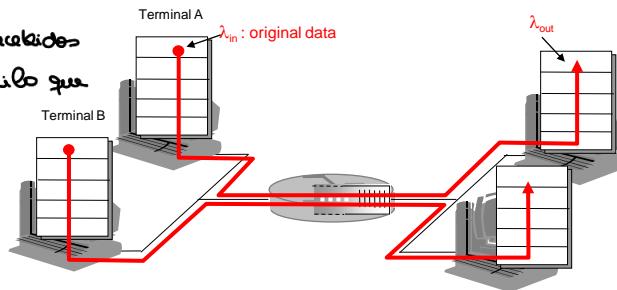
- λ_{in} → rate at which an application sends data (no header or retransmissions) → Dados originais a serem transmitidos / Descartando cabeçalhos e retransmissões

- λ_{out} → goodput

- Data that the application receives by unit of time (no header or retransmissions)

Apenas dados Recebidos no ponto de vista daquilo que a aplicação tem:

Sem cabeçalhos e retransmissões



Se ocorrerem retransmissões, vamos estar a aumentar tempo que demora

a chegar os dados corretamente ao nível da aplicação. Para os mesmos dados

enviados, vamos receber todos os dados enunciados, mas vamos demorar mais tempo a receber-los porque houveram retransmissões → Impacto a nível temporal

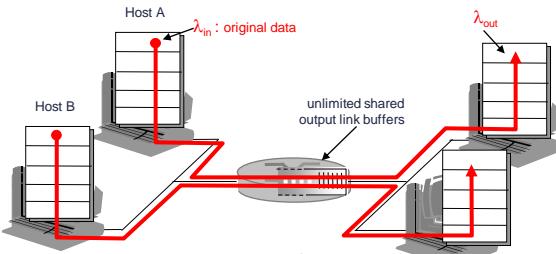
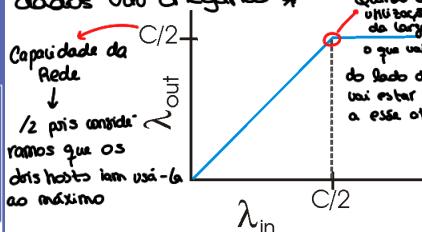
Se houver congestionamento na rede também há esse impacto, mesmo que não haja retransmissão

42

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission

Como o buffer do router é infinito, não há perdas, pois conforme dados vão chegando *



Quando chegar à utilização da metade da largura de banda o que vai chegar do lado do gráfico vai estar associado a esse outro

Não há perdas, há um atraso grande e acontece devido à congestão

- large delays when congested
- maximum achievable throughput

pris a capacidade máxima da rede foi atingida e os dados continuam a chegar, mas têm de ficar em espera no buffer

43

* Vão sendo colocados no buffer sem qualquer problema — Não é necessário haver retransmissão

43

Em relação ao goodput, quando atinge metade da largura de banda consegue ser constante, pois a capacidade da rede já foi atingida e portanto os dados passam a chegar ao receptor com uma frequência constante.

O único delay associado ao vai ser em relação ao tempo que estão presos no buffer

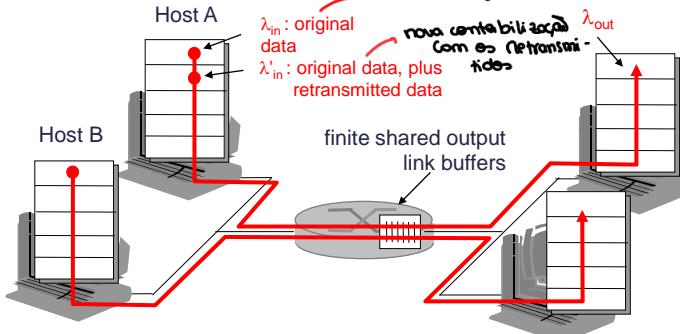
Causes/costs of congestion: scenario 2

- one router, finite buffers
- sender retransmission of lost packet

Como o buffer é finito, quando o buffer do router estiver cheio, os novos pacotes que lá chegam são perdidos

Tentar trazer retransmissão

Dados originais

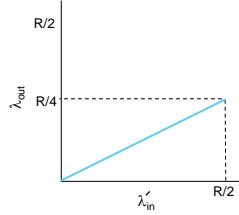
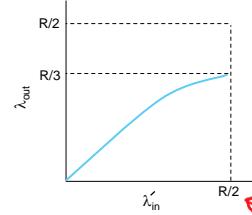
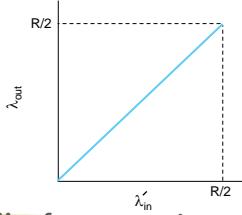


Novas contabilizações com os retransmitidos

44

Causes/costs of congestion: scenario 2

- “perfect” retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



costs of congestion:

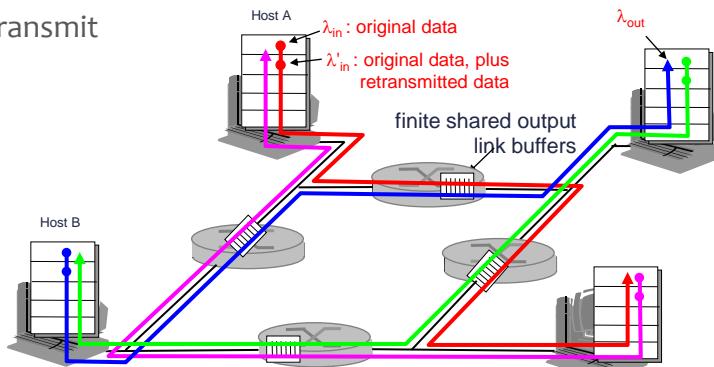
- more work (retrans) for given “goodput” → retransmissões para um determinado goodput
- unneeded retransmissions: link carries multiple copies of pkt

↳ retransmissões desnecessárias por termos múltiplas cópias do pacote que chegou lá na mesma

45

Causes/costs of congestion: scenario 3

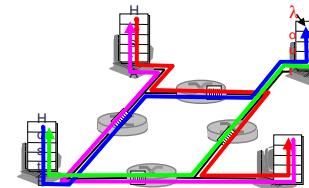
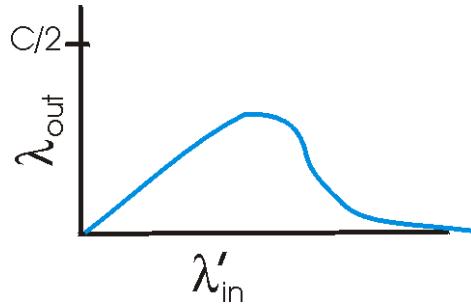
- four senders
- multihop paths
- timeout/retransmit



46

Causes/costs of congestion: scenario 3

- another “cost” of congestion:
 - when packet dropped, any “upstream transmission capacity used for that packet was wasted!



47

Congestion – λ_{out} decrease

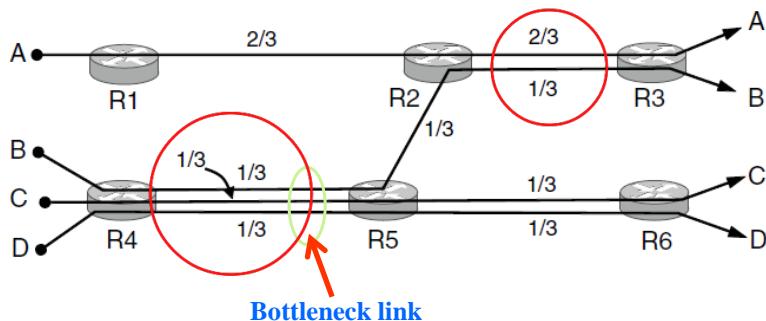
- Sharing connections
- Lost packets
 - Retransmitted packets
- Delayed packets
 - Timeout and retransmission
- With multiple hops: packet drop at last hop, all previous transmissions wasted

48

Desirable Bandwidth Allocation – Max-min fairness

Fair use gives bandwidth to all flows (no starvation)

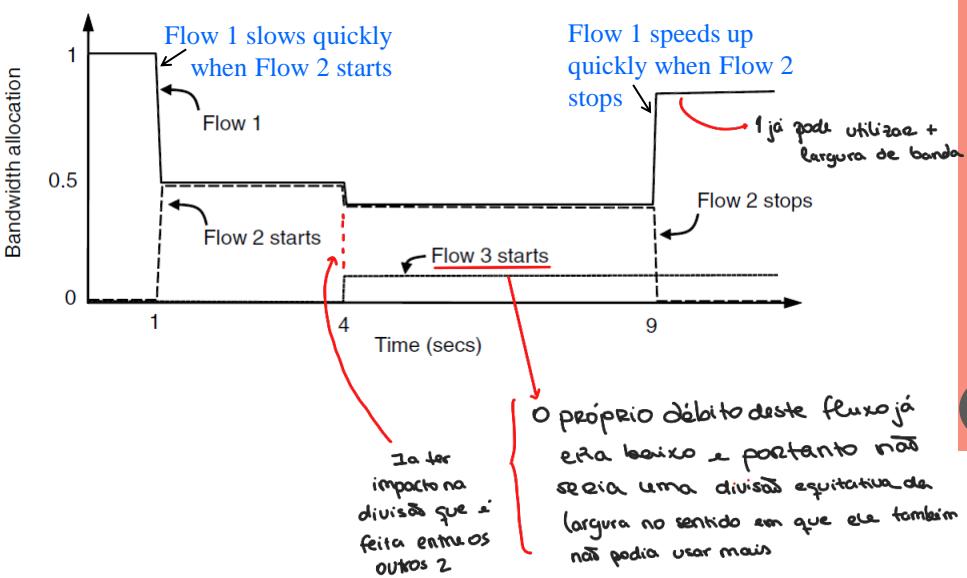
- Max-min fairness gives equal shares of bottleneck



49

Desirable Bandwidth Allocation – Bitrates along the time

Bitrates must converge quickly when traffic patterns change

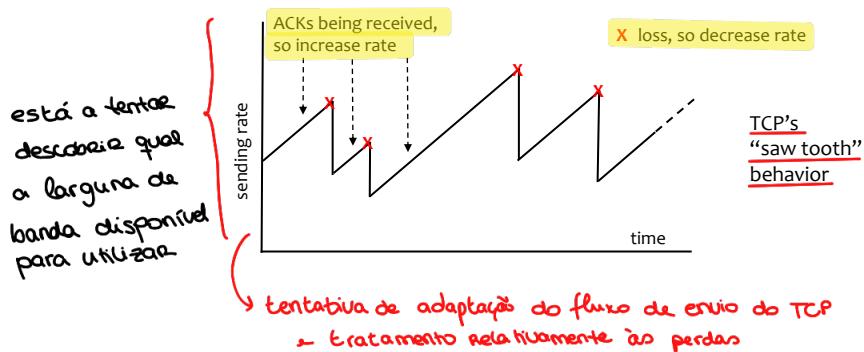


50

Ideia: procura para chegar a uma decisão equitativa → utilizável deste ponto de vista

TCP congestion control: bandwidth probing

- RFC 2581 - TCP Congestion Control
- “probing for bandwidth”: increase transmission rate on receipt of ACK, until eventually loss occurs, then decrease transmission rate
 - continue to increase on ACK, decrease on loss (since available bandwidth is changing, depending on other connections in network)



51

*por unidade de tempo visto que os envios dos ACK são periódicos

Additive Increase/Multiplicative Decrease

- Changes in channel capacity → adjustment of transmission rate
- New variable per connection → CongestionWindow
 - limits the amount of traffic in transit
 - $\text{MaxWin} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$
 - $\text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked})$
- Bitrate (byte/s) → $\frac{\text{CongestionWindow}}{\text{RTT}}$
 - $\text{tempo para recebermos um ACK}$
 - Também vai limitar o débito a que enviamos
 - Se não termos confirmações também não podemos enviar mais
- O que vamos ter na janela vai ser:
 - $\text{MaxWin} - \text{Bytes enviados} - \text{Bytes confirmados}$ ¹⁾
 - MaxWin é o tamanho total da nossa janela e depois temos a diferença do que é que já enviamos com o que é que já recebemos a confirmações
 - Esta medida que vamos recebendo confirmações vamos poder fazer o deslizar da janela e reduzir a diferença ¹⁾.

Additive Increase/Multiplicative Decrease

- Algorithm

- increases CongestionWindow by 1 segment**
 - for each RTT (Round Trip Time) → additive increase
- divide CongestionWindow by 2**
 - when there is a packet loss → multiplicative decrease

- In practice,

- Increases by ACK received
- MSS → Maximum Segment Size
- $Inc = MSS \times \frac{MSS}{CongestionWindow}$
- $CongestionWindow_{new} += Inc$

Máximo que podemos transportar na nossa rede ⇒ Nossa pacote máximo

53

TCP Congestion Control: more details

Segment loss event: reducing cwnd

- timeout: no response from receiver**
 - Corte é imediato e coloca cwnd a 1
 - cut cwnd to 1
- 3 duplicate ACKs: at least some segments getting through (recall fast retransmit)**
 - cut cwnd in half, less aggressively than on timeout

3 ACKs duplicados → alguns pacotes perdidos, mas não todos

Corte para metade

ACK received: increase cwnd

- Slow start phase:**
 - increase exponentially fast (despite name) at connection start, or following timeout
- congestion avoidance:**
 - increase linearly

Quando chegarmos a um determinado limite, deixaremos de aumentar exponencialmente e passaremos a aumentar linearmente (aumentamos mais lentamente) para não chegar a haver congestionamento.

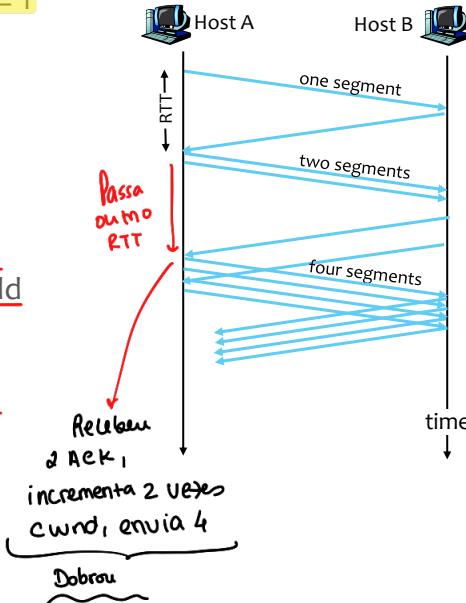
Tentativa de evitar a ocorrência de congestionamento devido ao envio de vários pacotes



54

TCP Slow Start

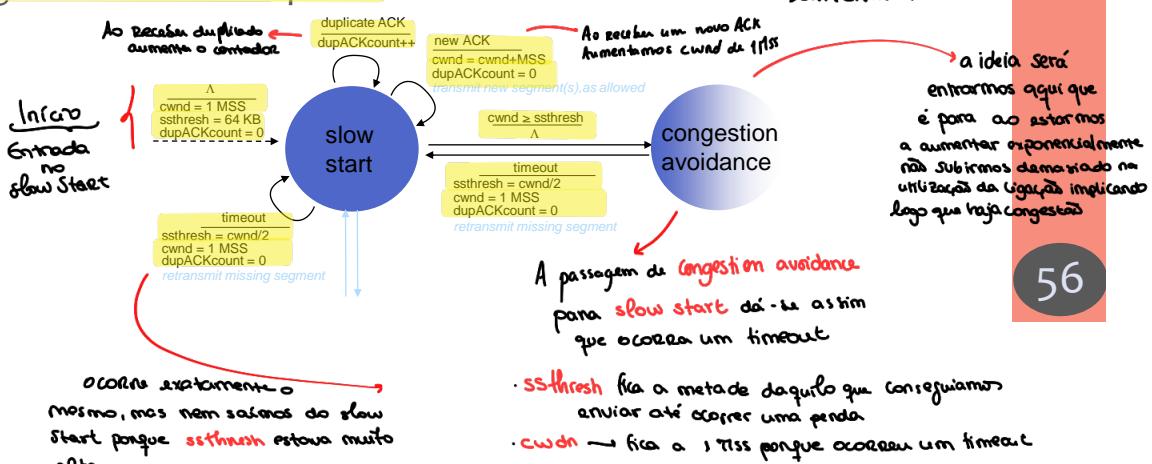
- when connection begins, cwnd = 1 MSS
- available bandwidth may be >> MSS/RTT
 - desirable to quickly ramp up to respectable rate
- increase rate exponentially until first loss event or when threshold reached
 - double cwnd every RTT
 - done by incrementing cwnd by 1 for every ACK received



55

Transitioning into/out of slowstart

- limita o tamanho da nossa janela*
- ssthresh: cwnd threshold maintained by TCP
 - on loss event (timeout): set ssthresh to cwnd/2
 - remember (half of) TCP rate when congestion last occurred
 - when cwnd >= ssthresh: transition from slowstart to congestion avoidance phase
 - onde vamos aumentar + lentamente cwnd



56

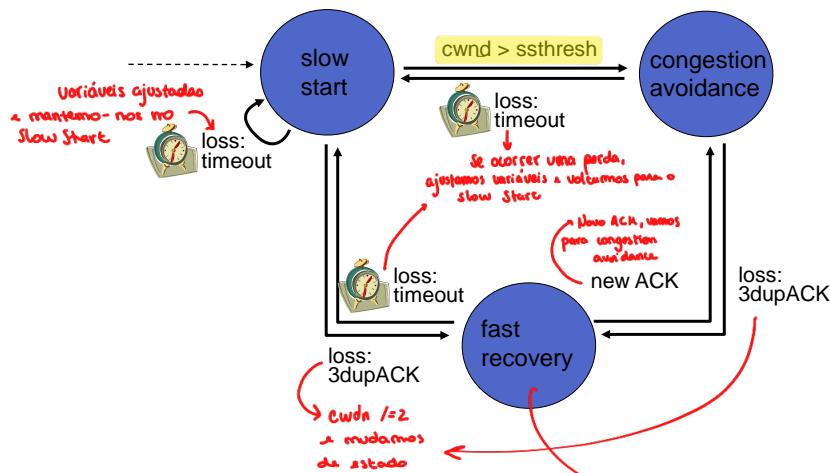
TCP: congestion avoidance

- when $cwnd > ssthresh$ grow $cwnd$ linearly
 - increase $cwnd$ by 1 MSS per RTT → em vez de aumentar por cada ACK recebido
 - approach possible congestion slower than in slowstart
 - implementation: $cwnd = cwnd + MSS * (MSS/cwnd)$ for each ACK received

57

TCP congestion control FSM: overview

Estados associados ao TCP

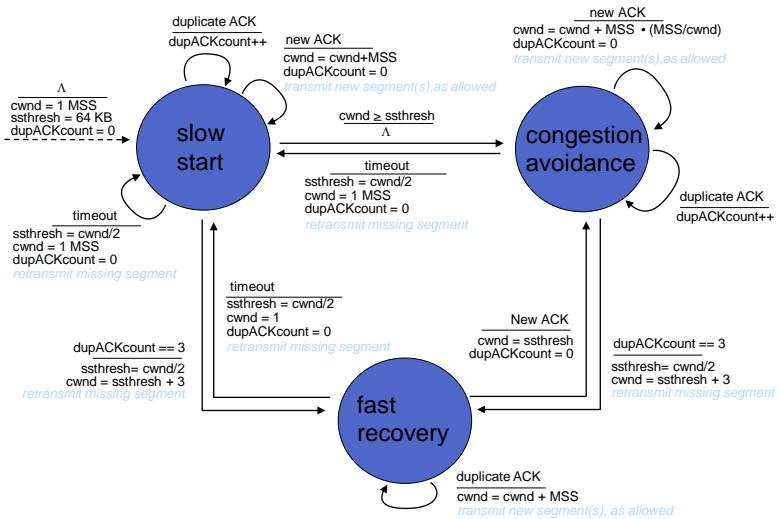


58

Compre logo o envio dos pacotes que estavam para trás sem esperar pelo timeout, quando se estavam a receber ACKs duplicados relativamente a pacotes que estavam a ser enviados "à frente do seu tempo", porque havia um buraco atrás.

RESUMO

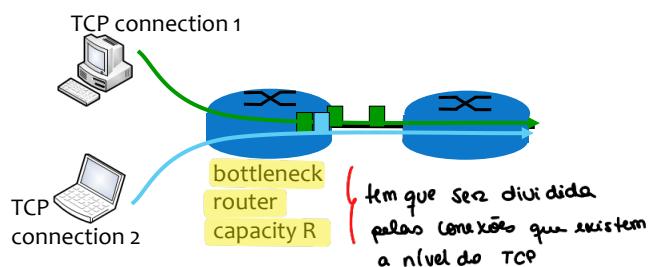
TCP congestion control FSM: details



59

TCP Fairness

fairness goal: if K TCP sessions share same bottleneck link of bandwidth R , each should have average rate of R/K

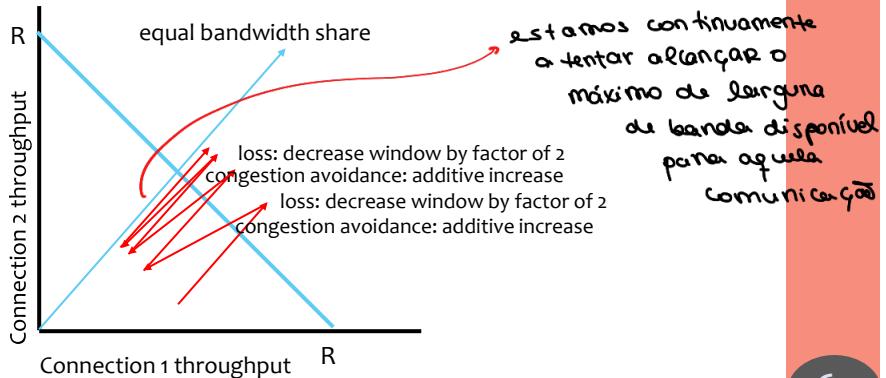


60

Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



61

Fairness (more)

Fairness and UDP

- multimedia apps often do not use TCP
 - do not want rate throttled by congestion control
- instead use UDP:
 - pump audio/video at constant rate, tolerate packet loss

• *aplicações multimédia*

↳ UDP + adequado

↳ Estas aplicações toleram + perdidas do que atrasos

Pretendem que hajam perdas + que se faça a recuperação do que hajam atrasos + não hajam perdas

De acordo com o seu débito de envio

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.

• web browsers do this

- example: link of rate R with already 9 TCP connections;

Divisão equitativa

- new app asks for 1 TCP, gets rate $R/10$
- new app asks for 11 TCPs, gets $\sim R/21$

$\frac{R}{21}$

• UDP tenta utilizar sempre o máximo de largura de banda disponível → se intercalarmos 1 UDP com TCP posteriormente ao TCP que chegou a um débito razoável, o UDP vai tentar usar o máximo de largura de banda disponível e o TCP com o mecanismo de controlo de congestão vai diminuir o seu débito, detectando as perdas e vai se adequando ao que é que é a utilização do canal de comunicação

62

Homework

1. Review slides
2. Read from Tanenbaum
 - o Chapter 6 – The Transport Layer

3. Answer questions at moodle

63