[Slide 1.1]

Os autores propoem uma nova abordagem, no ensino das redes, fundamentada no facto de a Internet ocupar cada vez mais o mundo das redes e no facto de serviços como Web, streaming de video, voz sobre IP, etc, terem tido um forte crescimento. Esta é uma abordagem Top-Down e é a abordagem que vamos seguir.

Abordagem Top-Down:
- Ênfase inicial à camada de aplicação e serviços, despertando o interesse do aluno.
- Sendo o mundo das redes vasto e complexo, envolvendo muitos conceitos, protocolos e tecnologias, esta abordagem suaviza a entrada neste mundo.

[Slide 1.2]

Vamos ver:
- o que é a Internet?
- o que é um protocolo?
- periferia da rede; hosts, rede de acesso, meio físico
- núcleo da rede: comutação de pacotes/circuitos, estrutura da Internet
- desempenho: perdas, atrasos, throughput
- camadas protocolares, modelos de serviço

Throughput (ou taxa de transferência) é a quantidade de dados transferidos de um lugar para outro, ou a quantidade de dados processados num determinado espaço de tempo. Pode usar-se o termo throughput para nos referirmos a quantidade de dados transferidos em discos rígidos ou numa rede, por exemplo, tendo como unidades básicas de medidas o Kbps, o Mbps ou o Gbps. O throughput pode também ser referido como taxa de transferência efetiva de um sistema.

[Slide 1.3]

Uma descrição do que é a Internet não é fácil. A Internet é uma miscelânea de coisas, é complexa e está sempre a mudar. Existem duas possibilidades:
- Descrever os seus componentes: hardware e software
- Descrever os serviços fornecidos, por esta infra-estrutura, às aplicações.

[Slide 1.4]

Descrição dos seus componentes:
- A Internet liga milhões de dispositivos chamados hosts ou end-systems (ex: PC, telemóvel, PDAs, frigoríficos, etc).
- Os hosts ou end-systems são os que correm/armazenam as aplicações.
- Os hosts conseguem comunicar através de canais de comunicação como fibra, cobre, etc. Estes diferentes canais conseguem transmitir a diferentes taxas/ritmos sendo esta medida em bits/segundo.
- Normalmente os end systems não estão ligados directamente. Estes estão ligados a dispositivos de comutação intermédios que conseguem transferir pacotes de um canal de comunicação para outro. Um pacote é um bocado de informação. Dispositivos de comutação mais conhecidos: i) routers, ii)

link-layer switches (comutadores da camada de ligação). Ambos transferem pacotes em direcção ao seu destinatário. Como? Veremos adiante.

Mais:
- Caminho: sequencia de canais de comunicação e dispositivos de comutação percorridos por um pacote.

[Slide 1.6]

Descrição dos seus componentes (continuação):

- Os end systems acedem à Internet atraves de ISPs (Internet Service Providers): i) ISPs residenciais como netcabo, sapo, etc.; ii) ISPs universitários como na UAlg; iii) ISPs que fornecem acesso wireless em aeroportos, cafes, etc.
- Cada ISP é um conjunto de canais de comunicação e dispositivos de comutação.
- ISPs podem fornecer vários tipos de acesso aos end systems: 56 kbps dial-up-Modem, cable modem, DSL, etc.
- Para acesso à Internet existem em todo o mundo ISPs de níveis superiores (nacionais e internacionais, ex: PortugalTelecom) que estão ligados por high speed routers e fibra óptica. Cada ISP é gerido de forma independente, corre o protocolo IP e respeita um conjunto de regras.

- Os elementos da Internet (end systems, comutadores de pacotes, etc) correm protocolos que controlam o envio e recepção de informação. O conjunto de protocolos principais usados na Internet é chamado TCP/IP. O seu nome vem dos 2 protocolos mais importantes: IP e TCP.
- Algumas organizações mantêm redes privadas, no sentido que os seus hosts não trocam mensagens com o exterior, chamadas de intranets. As intranets usam a mesma tecnologia (protocolos, links, routers, etc) da Internet. O acesso ao exterior é controlado com firewalls.
- Dada a importância dos protocolos na Internet é necessário que todos os elementos cumpram as regras e usem protocolos que fazem o que é perceptível pelos outros. Isto é definido pelos standards.
- Os RFCs (Request For Comments) são um conjunto de documentos de referência junto da comunidade Internet que descrevem, especificam, standardizam, debatem a estimulam a aplicação das normas/padrões de tecnologias e protocolos ligados à Internet e às redes em geral.

[Slide 1.7]

Descrição de serviços fornecidos:

- Se olharmos não para os componentes da Internet mas sim para os serviços fornecidos vemos que a Internet:
  -- Permite que aplicações a correr em diferentes hosts possam trocar informação. Estas aplicações usam os serviços de comunicação fornecidos pela Internet.
  -- Permite escolher opções de serviço: i) um serviço orientado à conexão (com ligação) que garante que todos os dados são entregues e na ordem adequada, e um serviço não orientado à conexão (sem ligação) em que não são dadas garantias de entrega dos dados. Uma aplicação faz uso de um

destes serviços e não de ambos. Não existem serviços que forneçam garantias de atraso.

- Os hosts na periferia utilizam serviços da Internet para trocar informação. Os links, routers, etc, fornecem os meios para transportar esta informação entre os programas nos end systems. Mas quais as características dos serviços fornecidos pela Internet? Alguém que desenvolva uma aplicação tem que usar um destes serviços. Eles são dois e ambos têm como objectivo a transferência de dados entre os end systems:

  -- Serviço orientado à conexão: Os programas cliente e servidor nos end systems trocam pacotes de controlo (chamado handshaking) antes da transferência dos dados. Após o handshaking está estabelecida uma conexão. O nome orientado à conexão significa que apenas os end systems estão cientes da conexão estabelecida. Os routers, switches não têm qualquer ideia desta conexão. De facto uma conexão não é mais que buffers e variaveis de estado nos end systems.

  O serviço orientado à conexão na Internet é o TCP que trás incluído: entrega fiável do fluxo de bytes (garante entrega sem erros e por ordem, confirmações (acks) e retransmissões em caso de perda), controlo de fluxo (o emissor não ultrapassa a capacidade do receptor; tem a ver com os hosts) e controlo de congestionamento (o emissor reduz o ritmo de transmissão em caso de congestionamento da rede; tem a ver com a rede). Note-se que para ser um serviço orientado à conexão basta fazer o handshaking, isto é, o restante não é requisito para se considerar orientado à conexão.

  -- Serviço não orientado à conexão: Não existe handshaking. Quando um dos lados de uma aplicação quer enviar dados simplesmente envia. Como não existe handshaking os dados podem ser entregues quanto antes. O serviço não orientado à conexão da Internet é o UDP. Note-se que isto não impede a própria aplicação de manter determinada informação de controlo para esse efeito. A aplicação não pode é confiar no serviço UDP para entrega fiável.

[Slide 1.8]

Um protocolo de rede é semelhante a um protocolo humano só que as acções são feitas por componentes de hardware ou software.

Exemplos:
- os protocolos implementados em hardware que vêm nas placas/interfaces permitem que duas placas/interfaces controlem o fluxo de bits no meio físico usado. É assim que duas placas com velocidade diferentes conseguem trocar bits (a mais rápida diminui a sua velocidade).
- alguns protocolos supervisionam o congestionamento dos end systems e controlam a taxa a que os pacotes são transmitidos entre o emissor e o receptor.
- alguns protocolos nos routers determinam os melhores caminhos para os pacotes viajarem.

[Slide 1.9]

Outro protocolo entre humanos: protocolo usado para colocar perguntas numa sala de aula.

[Slide 1.11]

A estrutura de uma rede inclui a periferia, acesso (dispositivos e meio fisico de acesso) e nucleo.

Na periferia da rede:
- encontram-se os end systems/hosts com as suas aplicações. Aliás é por isto que se chamam end systems (estão na borda/fronteira) ou hosts (armazenam/correm aplicações). Incluem: PCs, Servidores (Web, Email, etc), elementos móveis (portáteis, PDAs,...)

- os hosts podem ser máquinas: clientes (PCs, etc; é comum terem IP dinâmico) e servidoras (máquinas mais poderosas; é comum terem IP fixo)

Do ponto de vista do software existe outra definição de cliente e servidor. Um programa cliente a correr num host solicita serviços a um programa servidor a correr noutro host. Este é o tipo de aplicação mais comum na Internet.

Nalgumas aplicações um programa pode actuar como cliente e servidor. No caso do uso de telefone pela Internet (Skype) não faz sentido dizer que um solicita serviço e o outro serve. Agem como peers (Exemplos: Skype para chamadas entre PCs; BitTorrent para partilha de ficheiros). Existem assim dois modelos: client/server e peer-to-peer.

[Slide 1.12]

A rede de acesso inclui os links e dispositivos que ligam os end systems ao edge router (primeiro router no caminho).

Existem 3 categorias de redes de acesso:
- Residenciais: Liga os end systems de nossas casas à rede (Internet), isto é, ao edge router.
- Empresariais: Liga end systems de uma empresa ou instituição à rede (Internet), isto é, ao edge router.
- Móveis: Liga end systems móveis à rede (Internet), isto é, ao edge router.

Vamos analisar a largura de banda na rede de acesso e se o acesso é partilhado ou dedicado.

[Slide 1.13]

DSL: permite falar ao telefone e estar na Internet porque existe um splitter que separa as gamas de frequências usadas para esses fins. O facto de se usar frequências acima dos 4 kHz significa que a distancia tem de ser limitada. A linha de telefone é a mesma. O que muda são os extremos com mecanismos de codificação sofisticados. A taxa real varia em função da distancia. Normalmente são as companhias telefónicas que disponibilizam.

[Slide 1.14]

Cable modem: Esta tecnologia utiliza as redes de transmissão de TV por cabo convencionais (chamadas de CATV - Community Antenna Television) para transmitir dados. Faz uso da porção de banda não utilizada pela TV a cabo. Utiliza uma topologia de rede partilhada, onde todos os utilizadores partilham a mesma largura de banda.

Podemos ver que no cabo coaxial é usado FDM (Frequency Division Multiplexing) para alojar canais de video, de dados e de controlo.

Um splitter separa as frequências usadas para televisão e Internet.

[Slide 1.15]

HFC: O meio de transmissão inclui fibra e cabo coaxial. O meio é partilhado pelos utilizadores "pendurados" enquanto no DSL a ligacao é dedicada até à central. A taxa diminui quando estão muitos utilizadores "pendurados" e é muito alta quando estão poucos. É necessário um protocolo de acesso ao meio para evitar ou minimizar colisões (veremos adiante).

Numa rede por cabo teremos num extremo o central office, que fornece o serviço, de onde sai um cabo coaxial central (rede de distribuiçao). As residencias podem ligar-se a esta linha de distribuição através de Taps que são connectores em forma de T.

Como é a instalação nas residencias? Um splitter separa as frequências usadas para televisão e Internet. O sinal, com as frequencias usadas para Internet, é recebido por um cable modem que separa a rede de cabo coaxial da LAN (Local Area Network) do cliente.

[Slide 1.16]

Muitas casas começam a misturar acesso DSL ou cabo com uma LAN wireless de baixo custo. Este exemplo inclui:
- PCs (wireless e fixos)
- Um access point, que comunica com os PCs wireless
- Um router ao qual estão ligados o access point e os PCs fixos (ligação cabelada)
- Um modem cabo, para acesso à Internet, ao qual está ligado o router (ligação cabelada)

O router pode servir de firewall e implementar NAT. NAT permite que pacotes de uma máquina com endereço privado saiam para fora da rede com um endereço roteável. Será visto em redes II.

[Slide 1.17]

Ethernet é uma tecnologia de interconexão para redes locais (LANs) baseada no envio de pacotes. Define camada física e como é feito o acesso ao meio. A Ethernet foi padronizada pelo IEEE como 802.3. Existem standards Ethernet para par de fios (UTP, 4 pares), cabo coaxial e fibra, e para velocidade de 10 Mbps, 100 Mbps, 1Gbps, 10Gbps. Hoje já não se usa

o meio partilhado (coaxial) mas sim cabos UTP. Normalmente os PCs estão ligados por cabos UTP a um hub ou switch, e este está ligado a um router.

[Slide 1.18]

As wireless LANs que usam o protocolo 802.11b/g (Wi-Fi) são as que têm tido mais divulgação. Fornecem uma taxa de transmissão partilhada de 11 (802.11b) ou 54 Mbps (802.11g). O protocolo de acesso ao meio (MAC) no 802.11 é o CSMA/CA. O 802.11g é compatível com o 802.11b, o que significa que access points 802.11g irão comunicar com placas wireless 802.11b e vice versa. Será visto em redes II.

[Slide 1.19]

Atenção: apenas contabiliza atraso associado à colocação dos bits no canal. Não inclui atraso de propagação, processamento nos nós, múltiplos saltos, etc.

[Slide 1.20]

Um bit, ao viajar até ao destino, passa por muitos pares transmissor-recetor. O bit é enviado por ondas eletromagnéticas, ou pulsos ópticos, através de um meio físico. Os meios físicos podem ser diferentes nos vários links percorridos e incluem: pares trançados, fibra óptica, cabo coaxial, espectro de frequencias rádio.

Temos 2 tipos de meios: guiados (as ondas são guiadas num meio sólido) e não guiados (as ondas propagam na atmosfera ou no espaço, como na comunicação com satélites).

O meio físico é barato, comparando com outros custos, pelo que normalmente são colocados muitos cabos quando se abrem valas, mesmo que na altura não sejam todos usados. O custo de voltar a abrir as valas seria maior.

Nos cabos TP os pares de fios de cobre são entrelaçados. Usado também pelos telefones fixos. O entrelaçado permite reduzir interferências de outros fios.

Cabo directo: quando envolve switches/routers.
Cabo cruzado: entre PCs.
Nota: hoje os dispositivos sao inteligentes e podemos usar cabos diretos para tudo

[Slide 1.21]

Coaxial: é bidirecional (transmissao em ambos os sentidos), é um meio partilhado (todos os end systems recebem o que é enviado para o fio).

Bandabase/estreita: usado antigamente pelo standard Ethernet. Para sinais com gama de frequencias de 0 a MaxBandwidth.

Banda larga: usado hoje pelos fornecedores de Internet, permitindo vários canais. Note-se que banda larga é um conceito abstrato que define uma

conexão de dados em alta velocidade. Antigamente, banda larga significava uma banda base grande. Se queria transmitir mais dados, tinha que ter um espetro de frequências maior. Mas isso era porque as transmissões eram analógicas. Hoje, as transmissões são digitais e, neste caso, refere-se à existência de múltiplos canais.

Fibra óptica é um pedaço de vidro (ou material poliméricos) com capacidade de transmitir luz. Bit 0 = ausencia de luz; bit 1 = presença de luz.

[Slide 1.22]

Espectro electromagnético é o intervalo completo da radiação eletromagnética, que contém desde as ondas de rádio, as microondas, o infravermelho, a luz visível, os raios ultravioleta, os raios X, até aos radiação gama.

As microondas têm um comportamento diferentes das ondas rádio. Podem ser usadas para transmissão de dados mas têm problemas em ultrapassar obstáculos. Devido à sua frequência mais elevada podem transportar mais dados que as frequências rádio.

WiFi opera na banda dos 2.4 Ghz

[Slide 1.23]

As redes de telecomunicações dividem-se em:
- redes comutadas por circuitos: multiplexagem por divisão de frequência (FDM), multiplexagem por divisão de tempo (TDM), etc
- redes comutadas por pacotes: datagramas, circuitos virtuais (VCs)

Redes de datagramas: o endereço de destino num pacote determina o próximo nó. O caminho pode variar durante a sessão. Analogia: correios.

Redes de circuitos virtuais: cada pacote leva um identificador de circuito que determina o próximo nó. Caminho fixo determinado no estabelecimento da ligação. Ao contrário das redes de datagramas, são necessárias tabelas para manter o estado e informação dos circuitos virtuais. Pode permitir dar algumas garantias de qualidade de serviço.

[Slide 1.24]

O núcleo consiste numa malha de nós de encaminhamento (routers) que estão ligados entre si. Existem duas abordagens na construção do núcleo de uma rede:
- comutação de circuitos, em que é dedicado um circuito por ligação. Isto é, os recursos necessários num caminho para fornecer uma comunicação são reservados durante a duração da sessão de comunicação. Aqui o circuito é real. Quando é estabelecido um circuito é também reservada uma taxa de transmissão constante nos links utilizados durante a duração da sessão de comunicação. Como esta taxa foi reservada à priori o emissor pode emitir à taxa constante garantida.
- comutação de pacotes. Os recursos não são reservados à priori. Como não é reservada largura de banda, se a rede está congestionada os pacotes têm

de aguardar em queue. Fornece um serviço best-effort dado que vários utilizadores podem usar a largura de banda disponível.

Nem todas as redes podem ser classificadas em redes de circuitos ou de pacotes puras. Porque usa a Internet comutação de pacotes? Veremos adiante.

[Slide 1.25]
Os comutadores de pacotes usam normalmente transmissão store-and-forward em que o comutador tem de receber todo o pacote antes de poder transmitir o primeiro bit do pacote noutro link de saida. É introduzido um atraso store-and-forward proporcional ao tamanho do pacote.

Embora o packet switching seja mais eficiente no uso da largura de banda, tem atrasos difíceis de prever, o que é mau para algumas aplicações.

Na comutação de pacotes não existe:
- divisão da largura de banda em partes
- recursos dedicados
- reserva de recursos

Quanto tempo demora enviar um pacote de um host para outro atraves de uma rede comutada por pacotes? Vamos assumir para já que o atraso nas queues, de propagação e de processamento são negligenciáveis.

L = Tamanho do pacote
R = Taxa de transmissao, em bps, em cada um dos links
L/R = Tempo que demora a transmitir o pacote (colocar fora do dispositivo)
2L/R = Tempo que demora a transmitir o pacote nos dois links.
QL/R = Tempo que demora a transmitir o pacote em Q links.

Note-se que a recepção não é contabilizada porque o pacote não vai ser colocado no link de saida. Se não existisse store and forward, o fluxo de bits entrava e saía dos nós intermédios sem qualquer atraso de transmissão.

Nota: Nesta cadeira 1K nunca são 1024! Mas sim 1000! 1024 é nas cadeiras de programação.

[Slide 1.26]

Esta figura mostra uma rede de pacotes simples. Dois hosts (host A e B) estão ligados a um router e podem transmitir a 100 Mbps. Este, por sua vez, liga-se a outro router.

Neste exemplo todos os pacotes têm o mesmo tamanho mas nas redes Ethernet os pacotes têm tamanho mínimo de 64 bytes e máximo de pouco mais que 1500 bytes... veremos posteriormente.

Pode existir congestionamento se:
- existe uma ligação ao router (hosts estariam ligados a um switch, por exemplo, que se liga ao router) mas a velocidade de processamento do router impede que pacotes saiam à mesma velocidade com que chegam.

- existem várias ligações ao router que, em conjunto, excedem a capacidade do link de saída.

Ao contrário de TDM, em que cada host tem o mesmo slot em cada frame, aqui é diferente. Existe multiplexagem estatística: a sequencia de pacotes de A e B não tem um padrão fixo e, por isso, quem transmite mais tem mais probabilidade de ter dados à saída.

[Slide 1.27]

Existem duas funções chave no núcleo da rede:
- forwarding: quando chega um pacote é necessário mover o pacote do link/physical port de input para o link/physical port de output. Isto é feito com base em tabelas com determinadas informações. Acção local.
- routing: será necessário determinar os caminhos para os pacotes (preencher as tabelas). Isto é feito por algoritmos de routing. Acção global ao nível de toda a rede.

[Slide 1.28]

Na comutação por circuitos uma conexão tem recursos dedicados. A capacidade de um link depende da capacidade de comutação dos extremos

[Slide 1.29]

Na comutação de circuitos:
- se um link suporta n ligações então uma ligação entre dois hosts obtem 1/n da largura de banda do link
- várias conexões podem estar num mesmo link. Como? A largura de banda de um link está dividida em bocados/partes. Essas são atribuídas a chamadas/conexões. Se as chamadas/conexões não usam recursos então existe desperdício. Como dividir em partes?
-- divisão na frequência
-- divisão no tempo

FDM - O espectro de frequências de um link é dividido em bandas e cada uma delas é atribuída a uma conexão.

TDM - O tempo é dividido em frames de duração fixa. Cada frame é dividida num numero de time slots. Quando é estabelecida uma conexão num link, a rede dedica um time slot em cada frame à conexão. Apenas essa conexão pode usar o time slot para transmissão de dados. Exemplo: standard SONET/SDH, adoptado pela industria das telecomunicações, usa TDM.

[Slide 1.30]

Vamos comparar comutação de pacotes e comutação de circuitos:

- Pacotes: Não indicado para aplicações de tempo real devido aos atrasos imprevisíveis (sobretudo de queues). Mas permite maior utilização da largura de banda, é mais barato, mais simples porque não exige estabelecer circuitos.

- Circuitos: O contrário da comutação de pacotes.

Como é que a largura de banda é mais utilizada nas redes comutadas por pacotes? Suponhamos que existem N utilizadores com periodos de inactividade e periodos de actividade. Durante o periodo de atividade transmitem a uma taxa constante de 100 kbps/s. O periodo de actividade é de 10%. Neste cenário:

- Na comutação de circuitos é necessário reservar um circuito de 100kbps para cada utilizador. Assim, 1Mbps/100kbps=10 utilizadores em simultaneo no máximo.

- Na comutação de pacotes trabalhamos com probabilidades. Para um cenário de 35 utilizadores o que podemos dizer é que:

  -- A probabilidade de estarem mais de 10 utilizadores activos é menor que 0.0004

  -- Quando existem 10 ou menos utilizadores em simultâneo, o que acontece com probabilidade 1-0.0004=0.9996, a taxa de chegada de dados ao link é igual ou menor que 1Mbps, a taxa máxima de saída do link.

A resposta à primeira questão "como foi obtido o valor 0.0004?" é: 0.0004 = 1 - sum_{k=0}^{10} { (combinações de 35, k a k) x (0.1)^k x (1-0.1)^{35-k} }

[Slide 1.31]

Será que a comutação de pacotes é sempre melhor?

- Bom para tráfego de rajada (tráfego imprevisível). Aspecto positivo.
- Existem atrasos e perdas de pacotes quando existe congestionamento. Aspecto negativo.

[Slides 1.32 até 1.41]

Vimos que os end systems acedem à internet via uma rede de acesso situada na periferia da Internet. A rede de acesso está ligada à Internet através de uma hierarquia de ISPs.

- Access ISPs (ISPs residenciais, universidades, empresas) estão mais abaixo nesta hierarquia
- Tier-1 ISPs estão no topo da hierarquia e são em baixo número. Um Tier-1 ISP acaba por ser uma rede, com links e routers, que está ligada a outras redes. Os seus links têm é uma velocidade elevada e os seus routers têm de conseguir despachar pacotes a uma taxa muito elevada. Os tier-1 ISPs estão ligados entre si, estão ligados a tier-2 ISPs, e têm cobertura internacional. Os tier-1 são conhecidos por Internet backbone. Exemplo em Portugal: PT.
- Os tier-2 ISPs têm uma cobertura regional e são clientes Tier-1 ISPs. Muitas universidades e instituições ligam as suas redes directamente a um Tier-1 ISP, ou a um Tier-2 ISP, tornando-se clientes destes ISPs. O fornecedor cobra um valor, que normalmente depende da taxa de transmissão do link que liga os dois. Os Tier-2 ISPs podem optar por ligar-se directamente a outros Tier-2 e assim o tráfego não passa pelo Tier-1 ISP.

Nota: Alguns fornecedores Tier-1 são também fornecedores Tier-2, vendendo acesso internet directamente aos utilizadores finais e a ISPs abaixo na hierarquia.

POP (Point Of Presence) - termo para uma central com equipamento de um ISP (muitas vezes chamado 'nó'); usada para ligar os clientes ao backbone.
IXP (Internet Exchange Point) - localização onde vários ISPs estão presentes, se ligam e partilham meios de comunicação. Porquê? reduz custos, melhora desempenho.

[Slide 1.42]

Vamos agora ver o que pode acontecer a uma pacote que viaja pela Internet.

[Slide 1.43]

O que pode acontecer aos pacotes quando estes fazem o seu percurso? Podem ser armazenados nos buffers/queues dos routers quando a taxa de chegada excede a capacidade do link de saida. Neste caso aguardam em queue pela sua vez de sair.

[Slide 1.44 e 1.45]

Temos 4 fontes de atraso:

1 - Processamento no nó. Para verificação de existência de erros nos bits e determinar o link de saida (~microsegundos)

2 - Atraso nas queues. Porque ficam a aguardar a sua vez de serem transmitidos; depende do congestionamento no router (micro a milisegundos)

3 - Atraso de transmissão. L/R (micro a milisegundos)

4 - Atraso de propagação. Tempo que um bit leva a propagar desde o início até ao fim do link. A velocidade de propagação depende do meio físico mas anda entre [2*10^8 metros/segundo; 3*10^8 metros/segundo]. O atraso de propagação é igual a tamanho do link/velocidade de propagacao no meio. Numa WAN, o atraso de propagação total anda à volta dos milisegundos.

O atraso associado a um salto terá as 4 componentes mencionadas atrás. Temos de multiplicar por N se existe N-1 routers entre a fonte e o destino.

Estes atrasos podem variar muito. Nalgumas situações o atraso de propagação pode ser negligenciável (rede universitária) enquanto noutras pode ser significativa (ligação via satélite).

[Slide 1.46]

O atraso mais complicado é o de queue pois varia de pacote para pacote. Ex: o primeiro pacote da fila não tem atraso, dado que não tem pacotes à sua frente, enquanto o 10º terá algum atraso (tempo necessário para os pacotes que estão à sua frente sairem).

O atraso de queue é considerado grande ou pequeno dependendo de: taxa de chegada de pacotes à queue, taxa de transmissão do link, comportamento da chegada dos pacotes (periódico, rajada, ...).

La/R dá-nos uma medida da intensidade de trafego. Esta fórmula tem um papel importante no dimensionamento das queues.

- Se La/R > 1 a taxa de chegada de bits excede o que é despachado e a queue vai crescendo para infinito. Isto nunca deverá acontecer numa rede bem dimensionada.
- Se La/R < 1, depende da natureza das chegadas:
  -- Se um pacote chega a cada L/R segundos, todo o pacote chega a uma queue vazia e não há atraso de queue.
  -- Se os pacotes chegam em burts mas periodicamente (ex: N pacotes a cada (L/R)xN segundos) então o primeiro não tem atraso, o segundo tem atraso L/R, o pacote n tem (n-1)L/R segundos de atraso. Na realidade as chegadas não são periódicas.

Na realidade existem perdas(drops) porque as queues não são ilimitadas.

A curva do atraso mostrada no gráfico tem como base tráfego com comportamento de Poisson. Neste caso considera-se que os atrasos são aceitáveis (acima disso não) para intensidades de tráfego até 0.6.

[Slide 1.47]

Trace route é um pequeno programa que pode ser instalado em qualquer PC. É um programa de diagnóstico. Quando o utilizador especifica um hostname/IP o programa envia vários pacotes especiais em direcao ao destino. Se existem N-1 routers intermedios entao a fonte envia 3xN pacotes endereçados ao destino. O primeiro router retorna os 3 primeiros pacotes à fonte, o segundo retorna os 3 seguintes, etc (veremos como quando falarmos do ICMP e da camada de rede). A fonte regista os tempos que decorre desde o envio até obter cada uma das respostas, sabendo assim os atrasos na rede.

[Slide 1.48]

Nesta informação dada pelo traceroute temos em cada linha o número do router no caminho, o seu nome e número IP, e no fim os 3 atrasos obtidos.

[Slide 1.50]

Se o ficheiro a transferir possui F bits e levou T segundos a transferir, então o throughput médio da transferencia será F/T bits por segundo.
Aplicações como telefone e real-time video: querem baixo atraso e throughput consistente acima de um threshold.
Aplicações como transferencia de ficheiros: atraso não é crítico.

Podemos pensar nos bits como sendo um fluxo e os links como condutas.

[Slide 1.51]

Note-se que o servidor não pode enviar bits a uma taxa maior que Rs bits/sec e o router não pode encaminhar bits a uma taxa maior que Rc bits/sec. O end-to-end throughput é min{Rc,Rs}, link de bottleneck.

Exemplo: Esté a ser feito um download de um ficheiro MP3 de F=32 milhões de bits, o servidor tem uma taxa de transmissão de Rs=2Mbps e o utilizador tem um link de acesso de Rc=1Mbps. O tempo necessário para transferir o ficheiro será de 32 segundos=F/min{2Mbps,1Mbps}.

[Slide 1.58]

Exemplos de protocolos por camada:
- Aplicação: HTTP, SMTP, FTP, SDH, RTP, DNS, Telnet.
- Transporte: TCP, UDP, SCTP.
- Rede: IP, ICMP.
- Ligação: Ethernet 802.11 WiFi, FDDI, PPP.
- Física: RJ-45, RS-232, USB. (meios de conexão onde irão viajar os dados)

[Slide 1.59]

- mensagem é um pacote da camada de aplicação
- segmento é um pacote da camada de transporte (inclui header colocado pela camada de transporte + mensagem da camada de aplicação)
- datagrama é um pacote da camada de rede (inclui header colocado pela camada de rede + segmento da camada de transporte)
- frame é um pacote da camada de ligação (inclui header colocado pela camada de ligação + datagrama da camada de rede)

Uma mensagem que sai da camada de aplicação no nó origem, segue para as camadas de transporte, rede, ligação e física (onde são adicionados os respectivos headers) do nó origem. Será agora uma frame. No destino a frame é recebida pela camada física, que envia para as camadas de ligação, rede, transporte e aplicação (os respectivos headers são extraídos em cada camada). No seu percurso a frame pode passar por switches e/ou routers. Os switches operam até à camada de ligação, isto é, a frame recebida pela camada física é enviada para a camada de ligação que extrai o header. Um novo header é construido e adicionado, seguindo a frame novamente para a camada física em direção ao próximo salto. O mecanismo é semelhante num router, mas estes operam até à camada de rede.

[Slide 2.2]

Vamos falar agora da camada de aplicação. Os processos que correm nessa camada são específicos da aplicação. Exemplos de aplicações: facebook, youtube, skype, etc.

[Slide 2.3]

Objectivos do capítulo 2:

- Estudo de conceitos relacionados com aplicações em rede (como protocolos da camada de aplicação, interface com a camada de transporte, clientes e servidores, paradigma P2P)
- Desenvolvimento de aplicações: uma sobre TCP e outra sobre UDP (serviços da camada de transporte).

Vantagens de começar pela camada de aplicação:
- Muitas aplicações, que assentam nos protocolos que vamos estudar, são-nos familiares.
- Introduz muitos conceitos que serão vistos em profundidade nos próximos capítulos.

[Slide 2.4]

- Ligação remota: Telnet e SSH são dois exemplos.
- Partilha de ficheiros P2P: os programas nos hosts que participam na partilha de ficheiros serão semelhantes. Hosts vistos como peers.
- Aplicações com arquitectura client-servidor: email, web, transferência de ficheiros (ex: FTP), streaming de video.

Veremos email, Web, transferência de ficheiros, DNS. P2P, video streaming e voz sobre IP serão vistos em redes II.

[Slide 2.5]

Quando desenvolvemos aplicações em rede estamos a escrever programas que correm em diferentes sistemas e comunicam através de uma rede. Não estamos a falar de programas que correm em dispositivos, como routers ou switches, que estão no core da rede. Estes dispositivos não operam na camada de aplicação, operam sim nas camadas mais baixas (camada de rede e abaixo) e são da responsabilidade dos fabricantes.

Confinar o software da camada de aplicação aos hosts (end systems) fez com que aparecessem várias aplicações para a Internet num curto espaço de tempo.

[Slide 2.6]

Quando vamos desenvolver uma aplicação de rede temos primeiro que decidir qual o tipo de arquitectura da aplicação, isto é, como a aplicação está organizada pelos vários hosts (end systems). Note-se que não estamos a falar da arquitectura da rede (que fornece um conjunto de serviços às aplicações).

Arquitectura client-server inclui data centers e cloud computing:

- É dado o nome "data center" ou "server farm" a centrais de processamento de dados.

- É dado o nome "cloud computing" a um conjunto de serviços acessíveis pela internet que visam fornecer os mesmos serviços de um sistema operativo, ou parte deles. Consiste em compartilhar ferramentas que assentam na interligação de sistemas na internet em vez de se ter essas ferramentas localmente (mesmo nos servidores internos), tendo-se assim

acesso remoto, de qualquer lugar, por isso a alusão à nuvem. Neste contexto, o PC será visto apenas como um chip ligado à internet, a "grande nuvem" de computadores.

[Slide 2.7]

Numa arquitetura cliente-servidor existe um host servidor, que está sempre ligado e com um IP permanente, que serve os pedidos vindos dos hosts clientes. Assim os clientes conseguem sempre contactar os servidores. Os clientes não comunicam entre si (por exemplo, web browsers não comunicam directamente). Quando o número de clientes é elevado, podem ser usados clusters (chamados "server farm") para criar um server virtual potente.

[Slide 2.8]

Pares de hosts arbitrários (chamados peers) comunicam entre si directamente sem passar por um servidor. Exemplo: Gnutella, uma aplicação para partilha de ficheiros onde quaisquer hosts podem enviar, pedir, questionar onde está localizado um ficheiro, responder a questionários e encaminhar questionários. Neste caso milhares de peers podem participar na partilha de ficheiros. É escalonável porque apesar se um peer adicional significar mais pedidos, a capacidade de serviço e resposta a pedidos também aumenta.

[Slide 2.9]

Como comunicam as aplicações? Comunicam através de processos.

Comunicação inter-processo - as regras usadas na comunicação entre processos são definidas pelo sistema operativo. Não estamos interessados neste tipo de comunicação.

Qualquer sessão estabelecida para comunicação entre processos terá um processo chamado de cliente (o que inicia a comunicação) e outro chamado de servidor (que espera por ser contactado). Numa aplicação com arquitectura cliente-servidor como a Web, o browser é um processo cliente e o Web server é um processo servidor. Numa aplicação com arquitectura P2P como transferência de ficheiros, o host que faz o download (iniciou comunicação) tem o processo cliente e o host que faz o upload (foi contactado) tem o processo servidor. Note-se que um processo pode fazer upload e download e que uma aplicação pode consistir em vários pares de processos em comunicação (sessão). Numa sessão de comunicação um processo terá que ser chamado cliente e outro servidor, conforme o que foi dito atrás.

[Slide 2.10]

Os processos que trocam mensagens através da rede enviam/recebem mensagens através do seu socket (processo=casa; socket=porta de casa).

A figura ilustra uma comunicação entre dois processos através da Internet. O protocolo de transporte assumido é o TCP (o UDP também podia ser usado). O socket é o interface entre a camada de aplicação e a camada

de transporte. O socket está associado à API (application programming interface) entre a aplicação e a rede uma vez que é o interface com que as aplicações de rede são construídas. A pessoa que desenvolve a aplicação tem controlo na parte da camada de aplicação e pouco controlo na parte da camada de transporte. O único controlo que tem na parte da camada de transporte é a escolha do protocolo de transporte e definição de alguns parâmetros como "buffer máximo" e "tamanho máximo dos segmentos".

Socket = IP address + port number no contexto de uma conexão em particular ou estado de escuta. Um socket não é uma conexão, é sim o extremo de uma conexão.

Port number = identificador que define que serviço/aplicação

[Slide 2.11]

Quando é desenvolvida uma aplicação de rede é necessário atribuir-lhe um port number novo. Podem encontrar uma lista dos números de portos em www.iana.org.

O port number tem 16 bits: inteiro de 0 a 65535
-- 0-1023, well-known ports (serviços/aplicações essenciais, como mail, dns, etc)
-- 1024 a 49151, registered ports (oracle, bittorrent, etc; atribuidos e controlos pelo IANA)
-- 49152 a 65535, dynamic ports (atribuidos a processos cliente, como o desenvolvido nas aulas práticas; de atribuição dinamica pois aplicações como os nossos browsers nao estão sempre ativos)

[Slide 2.12]

RFCs são de domínio público. Ex: RFC 2616 refere-se ao HTTP, protocolo da Web. Se um programador desenvolver um browser de acordo com este RFC então o browser conseguirá obter páginas Web de qualquer servidor Web que siga também este RFC.

Alguns protocolos são proprietários e não estão intencionalmente disponíveis. Ex: Kazaa é o software P2P preferido a nível mundial para partilha de ficheiros.

Convem distinguir aplicação de rede e protocolo da camada de aplicação. Uma aplicação de rede tem várias componentes. No caso da Web: standard para formato do documento (HTML), Web browser (ex:Netscape), Web server (ex: Apache) e um protocolo da camada de aplicação que neste caso é o HTTP. O HTTP define o formato e sequência de mensagens que têm que ser trocadas entre o cliente e o servidor. Logo, o HTTP é apenas uma parte da chamada aplicação Web.

[Slide 2.13]

Ao desenvolver-se uma aplicação torna-se necessário escolher um dos protocolos da camada de transporte disponíveis. Como? Analisa-se os

serviços que cada um dos protocolos disponibiliza e escolhe-se o que fornece serviços mais adequados à aplicação em causa.

Os serviços que uma aplicação necessita, fornecidos pelo protocolo da camada de transporte, podem classificar-se em:

- Fiabilidade na transferência dos dados (ou data loss). Em audio ou video (real-time ou stored) uma pequena variação não tem efeitos graves. O email, transferência de ficheiros, Web, etc é exigido 100\% de fiabilidade.

- Largura de banda. Algumas aplicações têm de transmitir a uma determinada velocidade (ex: telefone pela Internet que codifica voz a 32 kbps; se não tiver esta taxa de transmissão pode mudar de codificação, para obter a taxa necessária, ou desiste). Outras aplicações são elásticas como o email, transferência de ficheiros, Web.

- Atraso. Aplicações em tempo real, como o telefone pela Internet, teleconferência, jogos multi-jogador, têm restrições de tempo apertadas na entrega dos dados (algumas centenas de milissegundos ou menos).

- Segurança. Um protocolo de transporte pode fornecer segurança encriptando todos os dados enviados que, na recepção, serão desencriptados. Pode ainda garantir a integridade dos dados.

[Slide 2.14]

Esta tabela sumariza as exigências de fiabilidade, largura de banda e atraso nalgumas aplicações conhecidas.

[Slide 2.15]

Nas redes TCP/IP estão disponíveis dois protocolos de transporte (TCP e UDP) e quando se cria uma aplicação uma das primeiras opções a tomar é qual deles usar. Os serviços disponibilizados pelo TCP são:

- Serviço orientado à conexão: procedimento de handshake que consiste na troca de informação de controlo antes de serem trocados os dados/mensagens da camada de aplicação. Após este processo é dito existir uma conexão entre os sockets dos dois processos. A conexão é full-duplex porque os dois processos podem enviar mensagens ao mesmo tempo. Diz-se orientado à conexão, e não conexão, porque a conexão é virtual.
- Serviço de transporte fiável: Os processos podem confiar no TCP para entregar os dados sem erros e na ordem certa. O fluxo de bytes enviado por um dos processos terá que ser idêntico ao recebido pelo outro processo.
- Mecanismo de controlo de fluxo: Faz com que um processo abrande o envio de dados para que o receptor não “transborde”.
- Mecanismo de controlo de congestionamento: Faz com que um processo abrande o envio de dados quando a rede entre o emissor e o receptor está congestionada. Regula também o uso de largura de banda por uma conexão TCP. Este aspecto é prejudicial, por exemplo, para aplicações de video e audio em tempo real que exigem um mínimo de largura de banda (além disso

são toleráveis perdas não necessitando de um serviço fiável). Aplicações em tempo real correm sobre UDP.
- Não dá garantias de atraso e de taxa de transmissão.

O UDP é um protocolo de transporte "leve" com um modelo de serviço minimalista. Não existe handshake (connectionless), não é fiável (não há garantia de que uma mensagem enviada para o socket chegue ao processo de destino; mensagens podem chegar ao processo de destino fora de ordem). Não tem mecanismo de controlo de fluxo e de congestionamento. Não tem garantias de atraso (como o TCP). Usado pelas aplicações em real-time já que estas toleram perdas.

[Slide 2.16]

A tabela mostra algumas aplicações e protocolo de transporte usado por estas.

O facto de se dizer que o TCP e UDP não oferecem garantias de atraso não significa que não possam existir aplicações sensíveis aos atrasos na Internet. Podem ser usados alguns truques para fornecer garantias satisfatórias embora não possam garantir restrições apertadas de atraso quando o congestionamento é grande na rede.

[Slide 2.19]

HyperText Transfer Protocol (HTTP): protocolo da camada de aplicação usado pela Web. O HTTP é implementado usado um programa cliente e um programa servidor. Estes conversam entre si trocando mensagens HTTP, isto é, mensagens com uma determinada estrutura que são trocadas entre o cliente e o servidor de uma determinada maneira.

Terminologia:
- Um objecto é um ficheiro endereçável através de um URL único. Numa página Web com um ficheiro HTML base e 5 imagens referenciadas no texto HTML temos 6 objectos.
- Os objectos são referenciados por um URL, isto é, um URL está associado a um objecto.

[Slide 2.20]

Um browser implementa a parte cliente do HTTP (para além disso permite visualizar a página e tem características configuráveis) enquanto o servidor Web implementa a parte servidora do HTTP (para além disso arquiva objectos; ex: apache).

A figura mostra o funcionamento do protocolo. O host cliente envia um HTTP request e o servidor envia HTTP response que inclui o objecto solicitado.

[Slide 2.21]

HTTP não guarda informações relativas ao cliente, isto é, se por exemplo um cliente pedir a mesma informação num curto espaço de tempo o servidor

não responde dizendo que acabou de enviar os objectos; o servidor reenvia os objectos.

[Slide 2.22]

O HTTP pode utilizar conexões persistentes ou não persistentes (default é persistente).

[Slide 2.23]

Passos envolvidos na transferência de uma página Web para o caso de conexões não persistentes:

1. Cliente HTTP inicia conexão TCP com servidor HTTP em www.someSchool.edu no port 80. A porta 80 é a porta default para o HTTP. Servidor HTTP no host www.someSchool.edu aguarda por conexão TCP no porto 80. Aceita conexão, notifica cliente.
2. Cliente HTTP envia HTTP request message para o socket associado à conexão. Mensagem indica que cliente quer objecto associado ao URL www.someSchool.edu/someDepartment/home.index.
3. O processo servidor recebe a mensagem de request através do seu socket associado à conexão, extrai o objecto pedido do disco, insere o objecto numa mensagem de resposta HTTP, e envia a mensagem de resposta para o cliente via o seu socket.

[Slide 2.24]

Passos envolvidos na transferencia de uma pagina Web para o caso de conexões não persistentes (continuação):
4. Servidor HTTP encerra conexão TCP.
5. Cliente HTTP recebe mensagem de resposta contendo ficheiro html, mostra html. Percorre ficheiro html, encontra 10 referências a objectos jpeg.
6. São repetidos os passos 1-5 para cada um dos 10 objectos jpeg.

Notas:
O browser vai mostrando a Web page à medida que vai recebendo. Dois browsers diferentes podem interpretar uma pagina Web de maneiras diferentes. Isso nada tem a ver com o protocolo HTTP. Este apenas define a comunicação entre o processo HTTP cliente e o processo HTTP servidor.

Quantas conexões são geradas neste exemplo? 11 conexões TCP. Cada conexão transporta quantas mensagens? 1 de request e 1 de response.

Nos browsers modernos estas conexões podem ser feitas em série (uma após outra) ou em paralelo. É configurável.

[Slide 2.25]

O RTT inclui todas aquelas componentes de atraso que vimos: propagação, queueing, ...

Quando um utilizador faz o “click” num link, o browser inicia uma conexão TCP que começa por um “three-way handshake”: o cliente envia um pequeno

segmento TCP para o servidor (SYN), o server responde com um pequeno segmento TCP que também faz acknowledgment do que veio do cliente (SYN+ACK). Até agora temos um RTT. Finalmente o cliente faz também o acknowlegment ao que veio do servidor.

A "request message" HTTP é enviada em simultâneo com a 3ª parte do handshake. O servidor envia a "response message" HTTP depois de receber a request message, o que consome outro RTT.

[Slide 2.26]

Desvantagem do modo não persistente - Uma conexão por objecto pedido (implica alocação de buffers TCP e manter variaveis TCP tanto no cliente como no servidor). O Web server fica assim sobrecarregado e cada objecto implica 2 RTTs. Note-se no entanto que alguns browsers podem fazer as conexões em paralelo.

Modo persistente - Uma página/objeto Web ou múltiplas páginas/objetos Web podem ser enviadas(os) por uma conexão apenas (o cliente e servidor têm que ser os mesmos). O servidor Web só fecha a conexão após indicação do cliente ou timeout (decorrer de um intervalo de tempo).

Existem duas versões de conexões persistentes - com pipelining e sem pipelining (a referida no slide é com pipelining, o default para HTTP 1.1)

Persistente sem pipelining - Embora conexões persistentes sem pipelining (1 RTT por objecto referenciado) sejam uma melhoria sobre o modo não persistente (2 RTT por objecto referenciado): i) delay pode ser reduzido ainda mais com pipelining; e ii) após o servidor enviar o objecto a conexão fica idle (não faz nada) enquanto espera pelo proximo request.

Persistente com pipelining - O cliente faz o request assim que encontra uma referência a um objecto no texto HTML. Logo pode fazer request sem ter recebido a resposta ao request anterior. É possível gastar 1 RTT apenas para todos os objectos referenciados e a conexão TCP fica pouco tempo idle.

[Slide 2.27]

O HTTP é um protocolo de texto (ASCII) sendo fácil o utilizador falar directamente com o servidor web em modo texto. Para isso basta fazer uma ligação telnet para a porta 80 (isto é, estabelece-se uma conexão tcp para a porta 80). Em seguida pode lançar-se um request na linha de comando em ASCII. Uma transação GET/POST inclui:
- Comando com o objecto a obter e versão HTTP (do browser) como argumentos.
- Informação adicional (cabeçalho).
- Linha em branco que indica fim de cabeçalho
- Dados (usado com o POST; no GET está vazio)

Neste exemplo as linhas do cabeçalho significam:

- Host: Máquina onde reside o documento. Poderia pensar-se que esta informação não era necessária já que já está estabelecida uma conexão TCP. Esta é exigida, no entanto, por Web proxies (veremos adiante).
- User-agent: Tipo de browser que está a fazer o pedido ao servidor. Esta informação é necessária caso se queira enviar versões diferentes de um objecto a tipos diferentes de user agents (com um mesmo URL).
- Accept: especifica os media types que são aceites como resposta.
- Accept-language: Indica que o utilizador prefere receber uma versão numa determinada língua. Caso não seja referido é enviada uma versão default.
- Accept-Encoding: Indica que codificações são aceites na resposta.
- Accept-Charset: Indica que character sets são aceites na resposta.
- Keep-Alive: especifica uma solicitação de timeout, periodo que o servidor aceita manter aberta uma conexão idle. Depois deste periodo a conexão é fechada.
- Connection: Ao colocar-se 'close' estamos a solicitar que o servidor assinale que a conexão vai encerrar depois de responder ao pedido. Isto é, a conexão não deve ser considerada persistente. Se estiver 'keep-alive' está a ser solicitado ao servidor que a conexão seja persistente.

[Slide 2.28]

O formato da mensagem request segue o formato do exemplo anterior!

metodo/method - GET, POST, HEAD
URL - identifica o objecto pretendido
versão/version - versão do HTTP implementada pelo browser.

Diferenças entre o GET e o POST: O POST é usado, por exemplo, quando um utilizador faz uma pesquisa num browser, preenchendo um form. O utilizador está a pedir uma página ao servidor mas o conteúdo da página depende dos dados enviados nos campos do form. No campo Entity Body está a informação inserida no form. Esta informação é processada, no servidor, para que a página seja gerada e enviada a quem solicitou.

[Slide 2.29]

O método GET também pode fazer o que o POST faz. Os dados inseridos nos campos do form são colocados no URL. No exemplo, o script a correr é 'animalsearch' e os campos do form (input) são 'monkey' e 'banana'.

[Slide 2.30]

HEAD - Semelhante ao GET. O servidor responde com uma mensagem HTTP mas deixa o objecto solicitado fora da resposta. Usado para testes.

PUT - Permite ao utilizador fazer o upload de um objecto para um directório específico no web server.

DELETE - permite ao utilizador, ou aplicação, apagar um objecto no web server.

[Slide 2.31]

Status line - versão do protocolo, código de status e mensagem de status (200=OK).
Date - data e hora da criação da mensagem HTTP (não da criação do objecto pedido).
Server - tipo de servidor.
Last-modified - indica a hora e data da criação do objecto ou sua última modificação (crítico para caching de objectos, tanto no cliente como nos servidores de cache na rede (conhecidos por proxy servers). Veremos adiante.
ETag - entity tag associada ao objecto que vai na resposta. Útil para pedidos condicionais futuros, já que identifica unicamente o objecto (cliente pode usar 'If Match: ' no futuro).
Accept-ranges - No envio do objecto ao cliente podem existir interrupções e chegar apenas uma parte do objecto. Nestas situações o cliente HTTP pode solicitar parte do objecto (a que não chegou). Ao enviar este header o servidor está a permitir este tipo de solicitação parcial.
Content-length - número de bytes do objecto enviado.
Keep-alive – semelhante ao que foi dito para o GET (timeout indica tempo que o host aceita ter a conexão idle aberta, sem fazer o close, e max é o numero máximo de requests na conexão persistente).
Connection - keep-alive indica que a conexão foi mantida persistente pelo servidor (close significaria que a conexão foi encerrada)
Content-type - tipo do objecto (no exemplo é dito que o entity body é texto HTML).

[Slide 2.32]

Para o código 301, a linha no header seria do género: "Location: http://www.wikipedia.org/index.php"

[Slide 2.33]

Note-se que as linhas inseridas nas request e response messages, pelo browser, dependem do tipo e versão do browser, configuração do browser feita pelo utilizador, se o browser tem uma versão do objecto em cache (que pode estar fora de prazo), etc

[Slide 2.34-2.35]

Um Web site tem por vezes necessidade de identificar um utilizador para restringir o acesso do utilizador ou para fornecer conteudos em função da identificação do utilizador. Para este efeito o HTTP usa cookies. Estes permitem fazer o "track" (controlar percurso) de um utilizador. A tecnologia cookie tem 4 componentes.

No exemplo: O servidor responde ao browser de Susana incluindo no header da resposta HTTP o número de identificação "Set-cookie: 167853". Quando o browser recebe a resposta HTTP adiciona uma entrada no seu cookie file para esse número de identicação. Esta entrada inclui o nome do servidor e o número de identificação. Agora, sempre que Susana pede qualquer coisa a este servidor, usando uma request message, o seu browser consulta o cookie file, extrai o número de identificação para este site e coloca-o no header da request message "cookie: 167853". Desta forma o site consegue acompanhar/registar todo o percurso de Susana no site. Susana

poderá, por exemplo, fazer o pagamento de vários itens no fim da sessão (que incluiu vários acessos), o servidor pode fazer-lhe sugestões de compra conforme preferências anteriores, etc. No servidor terá que existir uma entrada na base de dados que poderá incluir, por exemplo: IP + user account + número de identificação.

[Slide 2.36]

Problemas dos cookies:
- Cada browser tem uma área de armazenamento para cookies diferente. Isto significa que um cookie ficará associado um computador, conta de utilizador e web browser. Uma pessoa que use múltiplas contas, computadores e browsers tem vários cookies.
- Múltiplos utilizadores que usem um mesmo computador e browser não são diferenciadas se não usarem diferentes user accounts.
- Os cookies são trocados entre o servidor e o browser usando sessões HTTP normais e, por isso, estão visíveis a todos utilizadores que tenham um sniffer (ex: wireshark) na rede. Assim pode apanhar-se informação de autenticação, emails, etc.
- Um atacante pode modificar o conteudo de um cookie. Se um cookie contém o valor a pagar, por exemplo, o atacante pode lá colocar um valor mais baixo. Chamado cookie poisoning. Exemplo:

```
    GET /store/buy.asp?checkout=yes HTTP/1.0
    Cookie: SESSIONID=570321ASDD23SA2321; BasketSize=2; Item1=2892;
Item2=3210; TotalPrice=7500;
```

Note-se que o HTTP é também usado para transferir ficheiros (ex: ficheiros XML) sem que as máquinas tenham sequer browsers. Também é usado por aplicações P2P como protocolo para transferir ficheiros.

[Slide 2.37]

Um proxy server mantém, em disco, uma cópia dos objetos solicitados recentemente. Um browser pode ser configurado de forma a que todos os HTTP requests sejam enviados primeiro ao proxy server. Se o proxy server tiver o objeto solicitado envia-o ao browser. Se não tiver o objeto, o proxy server abre uma conexão TCP para o servidor original e envia um HTTP request solicitando o objeto. O servidor original envia o objeto num HTTP response ao proxy server. Este armazena o objeto no seu disco e envia uma cópia ao browser cliente usando um HTTP response.

[Slide 2.39]

Exemplo:
 Exemplo:
- O router da rede institucional está ligado ao router na Internet por um link a 1.5 Mbps;
- Os servidores na Internet estão espalhados por todo o mundo;
- O tamanho médio de um objeto é 100000 bits;
- A taxa de pedidos dos browsers na instituição para os servidores na Internet é de 15 pedidos por segundo (15*100K=1.5 Mbps);
- A intensidade de tráfego na LAN: 15*100000/1000000000 = 0.0015.= 0.15%. Isto é, 15 pedidos vezes o tamanho médio de um objeto a dividir por 1 Gbps que é uma característica da LAN institucional.

- Utilização do access link: 15*100 000/1 500 000 = 100%. O último número é a capacidade do link a 1.5 Mbps.
- Para uma intensidade de tráfego de 0.0015 na LAN, os atrasos serão na ordem dos microsegundos.
- Quando uma intensidade de tráfego se aproxima de 1 (link de acesso) os atrasos tornam-se muito grandes passando a ser da ordem dos minutos. Isto é inaceitável! Considerou-se um atraso na Internet médio de 2 segundos.

[Slide 2.40]

Esta solução baixa a intensidade de tráfego no link de acesso para 15*100 000/150 000 000 = 0.01=1%, o que resulta em atrasos negligenciáveis (milisegundos) entre os dois routers. O atraso total será aproximadamente 2 segundos. Solução de elevado custo.

[Slide 2.42]

Considere-se a solução alternativa de instalar um Web cache na rede da instituição:
- Na prática, a fração de pedidos que são satisfeitos por uma cache variam de 0.2 a 0.7. Vamos assumir 0.4 para esta instituição.
- 40% dos pedidos serão satisfeitos quase de imediato (~10 milisegundos) pela cache.
- 60% dos pedidos serão satisfeitos pelos servidores na Internet.
- A intensidade de tráfego no link de acesso reduz para 0.9 Mbps porque (0.6*1500000 = 900000).
- O atraso médio total é aproximadamente 1.2 segundos.(0.6 * 2.01 + milisegundos).
- Esta solução dá um tempo médio de resposta menor que a primeira hipótese e é mais barata.

[Slide 2.43]

Embora fazer caching reduza o tempo de atraso, introduz um novo problema: O objeto em cache pode ficar desactualizado e diferente do objeto no server original. O HTTP tem um mecanismo que permite que a cache verifique se os seus objetos estão actualizados. Este mecanismo é chamado de 'conditional GET'. Uma mensagem de request HTTP é chamada conditional GET se: i) a mensagem de request usa o método GET e ii) inclui a linha ' If-Modified-Since:' no header.

Exemplo:
1 – Em resposta a um pedido de um browser, o proxy cache envia uma mensagem ao Web server:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

2 – O Web server responde com uma response message, que inclui o objeto:

```
HTTP/1.1 200 OK
Date: Mon, 7 Jul 2003 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Web, 2 Jul 2003 09:23:24
Content-Type: image/gif

(data data data ...)
```

3 – O proxy cache envia o objeto para o browser e coloca-o também em cache (juntamente com a data da ultima modificação).
4 – Uns dias mais tarde outro browser pede o mesmo objeto via proxy cache. O proxy cache envia um conditional GET ao server para verificar se o objeto que possui está actualizado:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-Modified-since: Web, 2 Jul 2003 09:23:24
```

5 – O server apenas envia o objeto se este for mais recente que a data enviada em If-Modified-since. Neste exemplo não houve alteração:

```
HTTP/1.1 304 Not Modified
Date: Mon, 14 Jul 2003 15:39:29
Server: Apache/1.3.0 (Unix)

(empty body)
```

[Slide 2.44]

Vamos ver os protocolos da camada de aplicação que permitem que usemos o email.

[Slides 2.45-2.47]

Podemos dizer que o sistema de email possui 3 componentes:
1. Agente do utilizador (user agent) – permite aos utilizadores ler, responder, enviar emails. Também chamados de “mail readers”.
2. Servidores de email – são o core da infra-estrutura de email. Cada utilizador possui uma caixa de email (mailbox). Este servidor é responsável pela autenticação do utilizador no acto de leitura do email. Um servidor de email tem que lidar com a possível falha de outro servidor de email. Exemplo: quando o servidor de email de Alice não consegue enviar um email ao servidor de email de Bob, o servidor de email de Alice guarda a mensagem numa queue e tenta mais tarde. Depois de várias tentativas (default: 30 minutos entre tentativas; uns dias de máximo de tentativas) a mensagem é apagada e o emissor é notificado.
3. SMTP – protocolo principal, da camada de aplicação, para uso do email.
-- Utiliza o TCP para transferir um email do servidor de email do emissor para o servidor de email do destino. A parte cliente do SMTP é o servidor de email do emissor e a parte servidora do SMTP reside no servidor de email do destino. Ambas as partes cliente e servidora do STMP correm em todos os servidores de email. Quando um servidor de email envia um email age como um cliente SMTP. Quando um servidor de email recebe um email age como servidor SMTP.
-- Tem algumas restrições arcaicas: header e body da mensagem têm de estar codificados em ASCII 7-bits. Veremos adiante.
-- Comandos são código ASCII e a resposta é um código e uma frase. Os comandos e linhas de resposta terminam com um CRLF.

[Slide 2.48]

Neste caso o servidor de SMTP cliente é o servidor da Alice. Note-se que o SMTP não utiliza servidores de email intermédios. Se o servidor da Alice está num ponto do mundo e o servidor do Bob noutro, a conexão TCP é

uma conexão directa entre estes servidores. Se o servidor de Bob está em baixo a mensagem permanece no servidor de Alice e aguarda nova tentativa.

Registo MX no DNS: quando o mail server de Alice quer entregar o email ao server do Bob, tem de resolver o nome empresa.com. Terá que ser feito um query (tipo MX) ao servidor de DNS do domínio empresa.com. O servidor DNS responde com a informação de quem é o servidor de email (seu número IP).

[Slide 2.49]

Como são transferidas as mensagens usando o SMTP?
- Primeiro o cliente SMTP (servidor de mail que envia) estabelece uma conexão TCP na porta 25 com o servidor SMTP (servidor de mail que recebe). Se estiver down o client tenta novamente.
- Depois ocorre um handshaking ao nível da camada de aplicação, isto é, o cliente e servidor SMTP "conversam" antes de transferir informação. Nesta fase o SMTP cliente indica qual o email da pessoa que gerou a mensagem e o email do destino.
- Depois o SMTP cliente envia a mensagem e confia no TCP para enviar a mensagem sem erros.
- Se tiver outra mensagem para enviar o SMTP cliente repete o processo; caso contrário fecha a conexão TCP (conexão persistente).

No exemplo, o SMTP cliente é crepes.fr e o SMTP servidor é hamburger.edu (identificam-se nas duas primeiras linhas). As linhas com "C:" referem-se às linhas enviadas pelo cliente, para o seu TCP socket, e as linhas com "S:" referem-se às linhas enviadas pelo servidor para o seu TCP socket. A linha com "." indica fim da mensagem. Cada reply do servidor inclui um código e uma mensagem (opcional). Se tiver outra mensagem para enviar então o cliente SMTP inicia o processo com uma nova linha "MAIL FROM:", "." indica o fim da mensagem e QUIT só depois de todas as mensagens terem sido enviadas.

O subject, to, from, etc, que vemos num email é colocado depois de DATA. Isto é, faz parte da mensagem a enviar.

[Slide 2.50]

Para dialogar directamente com o servidor de email pode fazer-se telnet para a porta 25. Isto é, estamos a estabelecer uma conexão TCP entre o host local e o servidor de email. Estamos a enviar um email sem termos um cliente de email. Não esquecer colocar CRLF + '.' + CRLF após DATA. Note-se que em 'MAIL FROM' é possivel que alguem envie um email em nome de outro.

[Slide 2.51]

SMTP e HTTP:
- Ambos transferem ficheiros de um host para outro. O HTTP transfere entre o servidor Web e o cliente Web, enquanto que o SMTP tranfere de um servidor de email para outro servidor de email.
- Ambos usam conexões persistentes (no caso do HTTP é opcional).
- Diferenças:

-- HTTP é um pull (puxar) protocol. Isto é, é feito o load da informação. SMTP é um push (empurrar) protocol. Isto é, um servidor de email envia um email para outro servidor de email. No primeiro caso a conexão TCP é iniciada pela máquina que quer receber o ficheiro, no segundo caso a conexão TCP é iniciada pela máquina que quer enviar o ficheiro.
-- O SMTP exige que o header e body estejam codificados em 7-bit ASCII. O HTTP não.
-- Outra diferença está na forma como são manipuladas mensagens com texto e imagens. O HTTP coloca cada objeto na sua mensagem HTTP response. O SMTP coloca todos os objetos, relativos a uma mensagem, numa só mensagem.

[Slide 2.52]

- O header e o body da mensagem estão separados por CRLF.
- Algumas linhas do header são opcionais (Subject:, ...) outras são obrigatórias (From:, To:)
- Note-se que estas linhas do header nada têm a ver com as usadas no handshake do protocolo SMTP, embora sejam parecidas. Estas linhas do header fazem parte da mensagem em si. São inseridas normalmente pelo cliente de email.

Nota: o header e body têm de estar codificados em ASCII 7-bits. Se enviarmos, por exemplo, dados multimédia (binário) pode acontecer que estes incluam uma sequencia de bits correspondente a '.', Carrige Return e Line Feed, que são usados pelo SMTP para indicar fim de mensagem. Para não haver confusão todos os dados têm de ser codificados conforme o seguinte exemplo:
- 3 bytes = 8bits + 8bits + 8bits
- 4 grupos de 6 bits. Cada um é convertido para símbolos BASE64 (não inclui '.', etc).
- Cada símbolo BASE64 é convertido para ASCII 7-bits
Desta forma é garantido que alguns caracteres, como o '.', não são enviados. No destino é feito o processo inverso.

[Slide 2.53]

- Até agora foi assumido que um utilizador liga-se ao servidor de email e lá utiliza um software para ver o seu correio electrónico (se fosse usado apenas o SMTP). Usar um cliente de email num PC tem, contudo, muitas vantagens: podemos visualizar ficheiros multimédia e attachments mais convenientemente, podemos gerir localmente o email, não é necessário estarmos sempre ligado à rede (apenas quando queremos consultar o servidor de email).
- Porque não instalar um servidor SMTP no host de um utilizador? Este teria de estar sempre ligado à internet para receber email que chega a qualquer altura. Um servidor de email que é mantido por um ISP (ex: universidade) é partilhado por vários utilizadores e está sempre ligado.
- Um agente de email não pode usar o SMTP para recolher o email do servidor. O SMTP é um protocolo de envio. Para esse efeito existem os protocolos de acesso ao mail (POP e IMAP).
- Note-se que o user agent continua a usar o SMTP para enviar emails.

[Slide 2.54]

POP3:
- Protocolo simples e limitado.
- Após o user agent do cliente estabelecer uma conexão TCP na porta 110, o POP3 passa por três fases: autorização, transação e actualização.
- Fase autorização: o user agent envia um username e password para autenticar o utilizador
- Fase transação: o user agent pode recolher mensagens, marcar mensagens para apagar, desmarcar mensagens, obter estatísticas, etc. Exemplos de comandos:
-- stat, dá estatísticas
-- list, mostra as mensagens numeradas e seu tamanho em bytes
-- retr msg#, recolhe mensagem com número msg#
-- dele msg#, marca mensagem para apagar
-- rset, desmarca mensagens marcadas para apagar
- Fase actualização: ocorre quando é feito o quit. Apaga mensagens marcadas e faz o log off do servidor de email.

Podem experimentar fazer: telnet mailserver 110.

[Slide 2.55]

- Os clientes de POP podem usar dois modos: "download and keep mail on server" and "download and delete mail from server". O exemplo anterior usa “download and delete”. Este tem o problema de o utilizador não poder ver o seu email em máquinas diferentes. O modo “download-and-keep” já permite manter uma cópia no servidor (comando RSET é enviado pelo cliente para dizer ao servidor que não deve apagar os emails marcados quando for feito o quit).
- Com o POP3 o utilizador pode mover emails para folders após puxar o email. Isto tem problemas para os utilizadores nómadas que preferiam manter uma estrutura hierárquica de folders no servidor, que pode ser acedida de qualquer máquina, e guardar mensagens nos folders no servidor. O POP3 não tem comandos para o utilizador criar folders e associar mensagens a folders. O IMAP já tem comandos que permitem que o utilizador crie folders e mova mensagens entre folders.
- O POP3 não mantém informação de estado entre várias sessoes. O IMAP mantém.
- O IMAP permite também recolher apenas alguns objetos dos emails. Isto é importante quando a largura de banda é pequena e queremos apenas ver, por exemplo, o sender ou subject evitando trazer attachs.

Note-se que POP3 e IMAP são protocolos ASCII e não seguros por default. Usa-se POP3S, IMAPS ou camada SSL/TSL para esse efeito.

Web mail:
- Neste caso o user agent é um browser Web e o utilizador comunica com a sua mailbox através do HTTP. É normalmente usado um servidor IMAP para ser dada a possibilidade de criar folders e organizar msgs em folders. Ou seja, o acesso aos folders e mensagens é fornecido através de scripts que correm num servidor HTTP, e estes scripts usam o protocolo IMAP para comunicar com o servidor IMAP.

[Slide 2.57]

Um endereço IP consiste em 4 bytes com uma estrutura hierárquica rígida. Hierárquico porque, ao lermos da esquerda para a direita, obtemos informação de rede e de host.

O DNS tem duas componentes: 1) uma base de dados distribuida implementada numa hierarquia de servidores de DNS; 2) protocolo da camada de aplicação que permite que hosts questionem a base de dados distribuida. Os servidores de DNS são na maioria das vezes máquinas UNIX a correr o software BIND. O DNS corre sobre UDP e usa a porta 53.

[Slide 2.58]

- O DNS é usado por muitas aplicações e por outros protocolos da camada de aplicação (ex: SMTP) para traduzir um hostname em endereço IP. Isto pode adicionar algum atraso. Felizmente o endereço IP está muitas vezes na cache de um servidor de DNS por perto, o que ajuda a reduzir o tráfego DNS na rede e o atraso médio associado ao DNS.

- Um host pode ter um nome canónico (ex: relay1.west-coast.enterprise.com) ou aliases (ex: www.enterprise.com). É possível questionar o servidor de DNS para saber-se qual o nome canónico de um nome aliase ou endereço IP. Veremos adiante.

- Se o servidor de email tem um nome muito complicado é desejável arranjar um nome simples. Exemplo: ncorreia@ualg.pt, embora o nome canónico do servidor de email possa ser outro como mozart.si.ualg.pt. O DNS pode ser invocado por um servidor de email (SMTP) para obtenção do IP de um outro servidor de email com quem quer contactar para enviar-lhe emails.

- Permite também distribuir o tráfego por vários servidores que podem estar em locais diferentes e com IPs diferentes. É possível um nome estar associado a vários IPs. Quando alguém questiona o servidor de DNS para traduzir o nome em IP, este responde com toda a gama de IPs mas vai rodando na ordem dos IPs entre cada resposta que dá. Como um browser envia o HTTP request para o primeiro IP da lista, o DNS acaba por estar a distribuir o tráfego por varios servidores.

- Um servidor de DNS com toda a informação é desejável? Um único ponto de falha pode crashar toda a Internet, um único servidor teria de lidar com todos os queries, nunca poderia ficar perto de todos os clientes que fazem queries e assim alguns queries teriam que viajar muito, teria uma base de dados enorme e actualizado com demasiada frequência para criar entrada para todos os nomes de hosts.

[Slide 2.59]

Existem 3 classes de servidores de DNS: root servers, Top Level Domain servers e authoritative servers:
- Root servers: têm informação sobre os IPs dos TLD servers. Existem 13, a maior parte deles na América do Norte. Na prática cada um deles é um cluster de servidores por questões de sobrevivencia a falhas e rapidez.

- Top Level Domain (TLD) servers: são responsáveis pelos domínios de topo como .com, .org, etc, e têm informação sobre os IPs dos Authoritative servers.
- Authoritative servers: têm informação sobre os IPs dos hosts de uma organização que quer colocar os seus hosts acessíveis na internet. A organização pode manter o ser servidor authoritative ou pode pagar a um fornecedor de serviços para armazenar os seus registos num authoritative DNS server do fornecedor.

Existem ainda os servidores locais ou "default name servers" que podem existir nas organizações e são os servidores que são questionados pelos hosts. Este age como proxy fazendo o forward do query para a hierarquia de servidores de DNS. Faz cache.

[Slide 2.61]

A FCCN mantém o(s) servidor(s) do .pt.

[Slide 2.63]

Neste exemplo: O host cis.poly.edu, que quer traduzir o nome gaia.cs.umass.edu em IP, está no mesmo domínio que o servidor DNS local dns.poly.edu (estão no domínio poly.edu). O cis.poly.edu envia o seu pedido ao dns.poly.edu. Neste caso, uma vez que os servidores locais estão interessados em manter informação em cache, cis.poly.edu delega no local DNS server a tarefa de resolver o assunto (chamado query recursivo). O dns.poly.edu questiona o servidor de raiz que responde com o IP de um servidor que poderá ajudar (servidor do domínio .edu). Neste caso o dns.poly.edu não delegou no servidor raiz (chamado query iterativo). Então dns.poly.edu, utilizando o IP que lhe foi fornecido, questiona o servidor de .edu. Este responde com o IP de um servidor que poderá ajudar (servidor do domínio umass.edu, que é a máquina dns.cs.umass.edu). Então dns.poly.edu, utilizando o IP que lhe foi fornecido, questiona o servidor de umass.edu. Este tem conhecimento de qual o IP de gaia.cs.umass.edu e envia essa informação a dns.poly.edu. O dns.poly.edu coloca em cache essa informação e envia resposta ao host que colocou a questão (cis.poly.edu).

Temos então 1 query recursivo (primeiro) e 3 queries não recursivos (iterativos).

[Slide 2.64]

Neste exemplo todos os queries são recursivos. Query viaja da seguinte forma: cis.poly.edu, dns.poly.edu, servidor DNS raiz, servidor de .edu, servidor de umass.edu. A resposta viaja no sentido inverso. Na practica o funcionamento do slide anterior é o normal (mais usado): o query do host para o local server é recursivo e os outros não.

[Slide 2.65]

O DNS explora o caching de forma a melhorar a performance a nível de atraso e reduzir o número de mensagens DNS a fazer ricochete na Internet.

Numa cadeia de queries quando um servidor de DNS recebe uma resposta (contendo por exemplo um mapeamento de um hostname para um numero IP) ele pode colocar em cache essa informação. Depois este servidor pode fornecer essa informação mesmo não sendo authoritative para esse hostname. Os servidores descartam a informação em cache após x tempo (normalmente 2 dias). Um servidor local de DNS pode também armazenar em cache os números IP de servidores TLD para que um servidor de DNS local possa fazer o bypass dos servidores de raiz. Os servidores de raiz estão armazenados num ficheiro (não é de atualização dinâmica). Exemplo do que teria esse ficheiro:

```
.                       5d23h48m47s IN NS  I.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  E.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  D.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  A.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  H.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  C.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  G.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  F.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  B.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  J.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  K.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  L.ROOT-SERVERS.NET.
.                       5d23h48m47s IN NS  M.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.     6d23h48m47s IN A  192.36.148.17
E.ROOT-SERVERS.NET.     6d23h48m47s IN A  192.203.230.10
D.ROOT-SERVERS.NET.     6d23h48m47s IN A  128.8.10.90
A.ROOT-SERVERS.NET.     6d23h48m47s IN A  198.41.0.4
H.ROOT-SERVERS.NET.     6d23h48m47s IN A  128.63.2.53
C.ROOT-SERVERS.NET.     6d23h48m47s IN A  192.33.4.12
G.ROOT-SERVERS.NET.     6d23h48m47s IN A  192.112.36.4
F.ROOT-SERVERS.NET.     6d23h48m47s IN A  192.5.5.241
B.ROOT-SERVERS.NET.     6d23h48m47s IN A  128.9.0.107
J.ROOT-SERVERS.NET.     6d23h48m47s IN A  198.41.0.10
K.ROOT-SERVERS.NET.     6d23h48m47s IN A  193.0.14.129
L.ROOT-SERVERS.NET.     6d23h48m47s IN A  198.32.64.12
M.ROOT-SERVERS.NET.     6d23h48m47s IN A  202.12.27.33
```

[Slide 2.66]

Exemplos de RRs (estariam num ficheiro específico em disco; apenas a segunda linha dos exemplos):

```
name  ttl   class rr    ip
joe         IN    A     192.168.254.3

name        ttl   class rr    name
example.com.      IN    NS    ns1.example.com


name  ttl   class rr    canonical name
www         IN    CNAME joe.example.com.

name        ttl   class rr    pref  name
```

```
example.com.           IN    MX    10    mail.example.com.

name        ttl   class rr    name-server email-addr  (sn ref ret ex min)
example.com.           IN    SOA   ns1.example.com.
hostmaster.example.com. (
                              2013020800 ; se = serial number
                              172800     ; ref = refresh = 2d
                              900        ; ret = update retry = 15m
                              1209600    ; ex = expiry = 2w
                              3600       ; min = minimum = 1h
                              )
```

[Slide 2.67]

Os únicos tipos de mensagens DNS são: query e reply. Estas têm o mesmo formato:
- Os primeiros 12 bytes são o header que inclui:
  -- Número de 16 bits que identifica o query. É copiado para o reply permitindo que o cliente associe um reply a um query.
  -- Flags indicam: i) se é um query (0) ou um reply (1); ii) se o DNS server é authoritative (1), no caso de um reply; iii) se o cliente (host ou servidor de nomes que envia query) deseja que o servidor DNS, que recebe o query, faça recursividade caso não tenha o registo; iv) se o DNS server suporta recursividade (1), no caso de um reply.
  -- 4 campos "number of" que indicam o número de ocorrencias nas 4 secções seguintes.
- 4 secções com informações.

(será visto com detalhe na aula prática)

[Slide 2.68]

- secção question: tem informação sobre o query que está a ser feito. Inclui o nome e tipo (A, PTR, MX, ...) do que está a ser questionado.
- secção answer: contem os RRs que respondem à questão. Pode inclui vários RRs como resposta visto que, por exemplo, um hostname pode ter vários IPs.
- authority section: authority records (do tipo NS) nesta seção. Indicam nomes de servidores authoritative.
- additional section: contém RRs que estão relacionados com o query, mas não são resposta à questão. Exemplo: num query MX a secção answer contém um RR com o hostname canónico do servidor de email, enquanto a secção additional contém um RR do tipo A fornecendo o endereço IP para o hostname canónico do servidor de email.

(será visto com detalhe na aula prática)

[Slide 2.69]

O termo "registrar" refere-se a empresa que presta esse serviço (registrar=escrivão). Em portugal a FCCN era responsável por domínios. Agora existem empresas com acordo estabelecido com FCCN e ICANN que

vendem registos abaixo de .pt e prestam serviços de resolucao de nomes, etc.

Possível resposta à questão colocada: Se Alice quiser ver a pagina www.networkutopia.com, o seu host envia um DNS query para o seu servidor local de DNS. Este contacta um TLD para o dominio .com (se não tiver em cache o IP de um TLD terá que questionar um root server). Este TLD tem os RRs tipo NS e A indicados no exemplo porque a empresa os inseriu na base de dados. O TLD envia um reply ao servidor local de DNS de Alice contendo estes dois RRs. O servidor local de DNS envia um query a 212.212.212.1 a pedir um registo do tipo A para www.networkutopia.com. Ao receber a resposta, o servidor local envia-a para o host de Alice. Assim o seu browser pode iniciar uma conexão TCP para o IP de www.networkutopia.com.

[Slide 3.1]

Camada de transporte: encontra-se entre a camada de aplicação e a camada de rede, e é uma peça fundamental nesta arquitectura de cinco camadas porque tem a tarefa de garantir comunicação entre processos de aplicações que correm em hosts diferentes.

[Slide 3.4]

Comunicação lógica: Um protocolo da camada de transporte fornece uma comunicação lógica entre processos que correm em hosts diferentes. Comunicação lógica significa que do ponto de vista das aplicações é como se estas estivessem directamente ligadas. Mas, na realidade os hosts podem estar fisicamente distantes, ligados por vários routers e vários tipos de links. As aplicações utilizam esta ligação lógica para enviar os seus dados, sem se preocuparem com os detalhes de como são entregues.

Na camada de aplicação falamos de mensagens. Na camada de transporte falamos em segmentos (estes são os pacotes da camada de transporte). Segmentos resultam da partição das mensagens em bocados e posterior adição de um header a cada bocado. Os segmentos são enviados para a camada de rede onde são encapsulados em pacotes da camada de rede chamados datagramas. A camada de rede não examina o segmento (e respectivo header). No receptor a camada de rede extrai os segmentos, que estão dentro de datagramas, e envia-os para a camada de transporte.

[Slide 3.5]

Camada de rede: fornece uma comunicação logica entre hosts. O protocolo usado na Internet é o IP.

Camada de transporte: fornece uma comunicação lógica entre processos.

Analogia: Imagine-se duas casas com 12 crianças cada. As 12 crianças de uma casa são primos das 12 crianças na outra casa e gostam de trocar correspondencia. Essa correspondencia é entregue pelos serviços de correios (equivalente ao protocolo IP). Acontece que em cada casa existe uma criança (Ann e Bill, cada um na sua casa) responsável por recolher cartas (dos que vivem na mesma casa) e entregar ao carteiro, e por receber as cartas entregues pelo carteiro, distribuindo-as pelas crianças

da casa. Amanhã se Ann e Bill (equivalente ao protocolo de transporte) são substituidos por outras duas crianças o serviço de transporte é outro.

[Slide 3.6]

Temos 2 protocolos de transporte diferentes na Internet que fornecem serviços diferentes: TCP ou UDP. Quando alguém desenha uma aplicação tem de escolher qual destes deverá ser usado pela aplicação.

O IP, protocolo da camada de rede, fornece serviço best-effort. Isto é, o IP faz um esforço para entregar os segmentos entre os hosts mas não dá garantias. Isto quer dizer que não garante a integridade dos segmentos, não garante a entrega de segmentos (se queues congestionadas estes podem ser desprezados), e não garante a entrega dos segmentos pela sua ordem porque podem eventualmente seguir caminhos diferentes ou os routers terem politicas de queues (ex: prioridades) que altere a ordem.

Tanto o TCP como o UDP têm como objectivo estender o serviço do IP de forma a que a entrega seja feita entre processos. Isto é chamado de multiplexagem e desmultiplexagem na camada de transporte. Também fornecem integridade dos dados colocando no header dos segmentos campos para detecção de erros.

O TCP, para além dos 2 serviços descritos também inclui: controlo de fluxo (ajuste à velocidade do receptor) e controlo de congestionamento (para evitar congestionar uma rede que já está congestionada).

[Slide 3.8]

Um processo pode ter vários sockets. A camada de transporte não entrega dados directamente aos processo mas sim ao socket correcto. Cada socket tem um identificador único.

No envio: São recolhidos dados, de diferentes sockets, e estes são partidos em pedaços. A cada pedaço é adicionada informação (header) necessária na recepção. Estes segmentos criados são enviados para a camada de rede.

Na recepção: Cada segmento tem um conjunto de campos (port origem, port destino, etc, que foi colocada no header pelo emissor) que permitem que a camada de transporte saiba para que socket deve encaminhar o segmento. Um segmento é entregue num socket.

[Slide 3.9]

Para além do port origem e port destino, os segmentos TCP e UDP têm outros campos de que falaremos adiante. Cada número de port tem 16 bits (0-65535). Os port numbers de 0 a 1023 são chamados “Well-known port numbers” e estão reservados para aplicações importantes e bem conhecidas (HTTP, FTP, …).

[Slide 3.10]

Para criar um socket UDP usamos algo do género do que se segue (podemos especificar um número de port específico ou deixar que seja atribuído um número aleatório de forma automática):

```
DatagramSocket mySocket = new DatagramSocket();
DatagramSocket mySocket = new DatagramSocket(19157);
```

Desta forma a camada de transporte atribui um número de port de forma automática (na gama 1024-65535). Pode colocar-se como parâmetro um número de port específico. Normalmente os servidores colocam um port específico enquanto os clientes pedem port de forma automática.

Nota: O código atrás é Java. Nas aulas práticas estamos a usar python. Aqui está a ser usado apenas para exemplificar.

[Slide 3.11]

O port origem e o port destino são ambos colocados no header de um segmento, quando este é criado. A camada de rede coloca o endereço IP origem e destino quando acrescenta o seu header (passa a ser um datagrama). Desta forma é possível enviar dados no sentido inverso. Exemplo: programa UDP, feito na aula prática, converte texto para maiusculas e devolve.

Pacotes com diferentes endereços IP fonte, e/ou diferentes números de porto fonte, são enviados para o mesmo socket de P1.

Resposta ao ? em cima: source port: 6428; dest. Port: 5775.
Resposta ao ? em baixo: source port: 5775; dest. Port: 6428.

[Slide 3.12]

Ao contrário do UDP, dois segmentos TCP com IPs fonte diferentes ou número de port diferentes serão direcionados para sockets diferentes numa máquina destino.

O server TCP tem um “welcoming socket”, que espera pedidos de clientes que querem estabelecer conexões TCP num port. O cliente envia um segmento de estabelecimento de conexão. Este é um segmento com um determinado bit (SYN) activado no header. Além disso cria um socket cliente cujo número vai no segmento dirigido ao server. Ao receber esse segmento o servidor cria um novo socket, identificado pelo 4-tuplo, que ficará associado a esta conexão TCP.

[Slide 3.13]

O cliente A e B escolhem números de socket de forma independente e, por isso, podem usar os mesmos números. É possível distinguir os segmentos de cada cliente porque terão IPs diferentes (no header do datagrama onde estão incluidos). A máquina B também consegue ter conexões em paralelo com a máquina C (números de socket diferentes).

Se fosse uma ligação sem conexão, os dois P3 poderiam enviar para um mesmo socket na máquina C (estão a usar o mesmo IP destino e socket

destino). Como é uma ligação com conexão isso não é possível. Só os datagramas com o mesmo (IP fonte, IP destino, socket fonte, socket destino) irão pertencer a uma mesma conexão que liga um socket na origem com um socket no destino (isto é, nenhum outro processo consegue enviar para estes sockets).

[Slide 3.14]

Os servidores Web ou produzem um processo para cada conexão (cliente) ou criam uma "thread" com um socket para cada conexão. Thread é uma forma de um processo se dividir em duas ou mais tarefas que podem ser executadas em simultâneo.

[Slide 3.16]

UDP apenas multiplexa/desmultiplexa e verifica erros (perante um erro faz drop). A aplicação quase que fala com o IP na camada de rede uma vez que pouco é acrescentado.

Estabelecer conexão exigiria manter informação de estado: buffers de envio e recepção, parâmetros de controlo de congestionamento, números de sequencia e acknowledgments.

Tamanho do header do UDP: 8 bytes (o TCP tem 20 bytes).

[Slide 3.17]

Exemplos de aplicações que usam UDP:
- DNS. Muitas aplicações (ex: web, mail, etc) fazer queries ao DNS para resolver nomes. Por isso terá que ser uma operação rápida, senão existe muito delay.
- RIP. Protocolo para actualização de tabelas de routing. Periodicamente os hosts que correm o RIP trocam informação. Se alguma actualização se perder não há problema porque virá outra actualização x tempo depois.
- SNMP. Protocolo usado para monitorar e controlar serviços e dispositivos de uma rede TCP/IP. Usado em situações em que a rede está com dificuldades, facto que dificulta o uso do protocolo TCP (para fornecer fiabilidade, controlo de fluxo e de congestionamento, fica mais pesado para a rede).

Note-se que nada impede que uma aplicação sobre UDP tenha fiabilidade desde que seja ela a gerir isso. A aplicação não pode é confiar na camada de transporte para esse efeito.

[Slide 3.18]

Origem: Todas as words de 16 bits do payload do segmento são adicionadas. Se existe overflow, o bit que sobra entra no outro lado (é adicionado). É feito o complemento a 1 do resultado (0s passam a 1s; 1s passam a 0s). Note-se que o payload é preenchido ('padded') com zeros de forma a garantir que existe um número inteiro de words. Neste caso o campo length no header indica o tamanho original (sem preenchimento).

Destino: Todas as words são adicionadas, incluindo o checksum. Se não existem erros dá tudo 1s (ou 0s se for feito o complemento a 1).

[Slide 3.21]

Vamos abordar os principios da transferencia de dados fiável, mas de forma geral. Isto porque estes principios também podem ser aplicados noutras camadas como a camada de ligação e camada de aplicação.

A figura mostra um processo emissor, na camada de aplicação, que envia dados para um tubo na camada de transporte. Este tubo representa um canal fiável. No outro lado do tubo saem dados que são enviados para a processo recetor na camada de aplicação.

Em abstrato, o que é fornecido à camada superior é um canal fiável através do qual os dados podem fluir de forma fiável (sem erros, sem perdas, e entregues na mesma ordem que foram enviados). Um reliable data tranfer (rdt) protocol tem esta responsabilidade.

[Slide 3.22]

O que irá dificultar a tarefa do canal fiável é o facto de na camada de rede abaixo (ex: IP) o canal não ser fiável (unreliable). O IP é então um unreliable data transfer (udt) protocol.

[Slide 3.23,3.24]

Na figura são mostradas interfaces/métodos, para transferencia dos dados, na fronteira entre a camada de aplicação e a camada de transporte, e entre a camada de transporte e a camada de rede:

rdt_send(): Usado pela camada acima (neste caso a camada de aplicação) para passar os dados ao canal reliable. Está a ser invocada a parte emissora do protocolo rdt. Usado no host emissor.

rdt_rcv(): Usado pela camada de baixo (neste caso a camada de rede) para passar os dados ao canal reliable. Está a ser invocada a parte receptora do protocolo rdt. Usado no host receptor quando um pacote chega pelo canal unreliable (ex: IP).

deliver_data(): Usado pela camada de transporte para entregar dados à camada superior.

udt_send(): Usado pela camada de transporte para envio de dados para o canal unreliable (neste caso a camada de redes).

[Slide 3.25]

Vamos apresentar em seguida versões de um reliable data transfer (rdt) protocol para as partes cliente e servidor, de forma a serem considerandos cada vez mais aspectos nas várias versões. Usaremos máquinas de estados finitos (finite state machine, FSM).

Nas FSM:

- estado: Representado por um círculo (com o nome do estado lá dentro)
- transição: Representado por uma seta que liga um estado a outro. Uma seta tem ao lado o evento que provocou a transição de um estado para outro, e ações a tomar ao transitar de estado (estas informações estão separadas por uma barra horizontal).

Nota:
- Quando ocorre um evento que provoca uma transição de estado mas não ocorre nenhuma acção, é usado o símbolo Lambda no sitio da ação.
- Quando ocorre um evento que não muda o estado, é usada uma seta para o próprio estado.

[Slide 3.26]

Versão 1.0: canal sem erros e sem perdas.

Emissor: Aguarda chamada da camada superior. Quando surge o evento rdt_send(), o protocolo terá de fazer a acção de criar um pacote com a função make_pkt e terá de enviá-lo para o canal unreliable (camada abaixo). Para um protocolo da camada de transporte isto significa criar segmentos com um determinado header. Em seguida continua a aguardar por nova chamada (fica no mesmo estado).

Receptor: O protocolo irá receber o pacote quando ocorre o evento rdt_pkt(), e a acção a tomar é extrair os dados do pacote (segmento) e enviá-los à camada superior usando a acção deliver_data(). Em seguida continua a aguardar por nova chegada de pacote (fica no mesmo estado).

[Slide 3.28]

Versão 2.0: Vamos para um cenário mais realista em que podem existir erros nos pacotes (durante a propagação, armazenamento nos buffers, etc). Ainda não são consideradas perdas. É necessário:
- Mecanismo para deteção de erros: checksum (veremos outras técnicas de deteção e correção de erros)
- Feedback do receptor: positive e negative acknowledgements.
- Retransmissão de pacote caso seja recebido um NAK (negative acknowledgement) pelo emissor.

[Slide 3.29]

Emissor: Neste caso, depois de fazer o pacote e enviá-lo, o protocolo passa para um novo estado em que fica a aguardar um ACK ou NAK. Se ocorreu o evento de chegada de pacote (rdt_rcv) e é um NAK, então ocorre a acção de voltar a enviar o pacote de dados. Se ocorreu o evento de chegada de pacote (rdt_rcv) e é um ACK, então não ocorre qualquer acção e o protocolo volta ao estado de espera de dados.

[Slide 3.30,3.31]

Mostra que evento, ações e transições de estado ocorrem num cenário sem erros e com erros, usando a versão 2.0 (slide 3.28).

[Slide 3.32]

A versão 2.0 do nosso protocolo tem um problema: não controla a existência de erros nos ACKs e NAKs. O emissor não sabe verdadeiramente o que aconteceu no destino. Uma possibilidade é:
- Colocar checksum nos ACKs e NAKs que o receptor enviar.
- Como o receptor não sabe se os seus ACKs ou NAKs chegaram bem ou mal, os pacotes enviados pelo emissor têm de vir numerados para que o receptor saiba se é um duplicado ou se é um novo pacote.

Esta nova versão será a 2.1.

[Slide 3.33]

O que faz o emissor na versão 2.1:
- numera os pacotes com 0 ou 1.
- envia pacote 0 ao destino e passa ao estado de espera de ACK ou NAK.
- se recebeu um acknowledgment (ACK ou NAK) para o pacote 0 e, ou recebeu NAK ou está corrompido, então reenvia o pacote de dados e mantem-se no mesmo estado.
- se recebeu um acknowledgment (ACK ou NAK) para o pacote 0, é um ACK e não está corrompido, então não faz nada e simplesmente passa para o estado de espera de novos dados da camada superior (pacote que será numerado com o número 1)
- faz o que foi dito atrás para o pacote com número 1.

[Slide 3.34]

O que faz o receptor: aguarda pacotes da camada inferior, que podem ter o número 0 ou 1.
- Se estava a aguardar pelo pacote 0:
 -- recebeu o pacote 0, e não estão corrompidos: i) extrai dados e envia para a camada superior; ii) cria um acknowledgment ACK e envia para o emissor; iii) muda de estado, ie, fica a aguardar o pacote 1.
 -- está corrompido: cria um acknowledgment NAK e envia para o emissor.
 -- recebeu o pacote 1 (pacote anterior), e não está corrompido, cria um acknowledgment ACK e envia para o emissor. Isto pode acontecer se o receptor mudou anteriormente de estado porque estava tudo ok mas o seu ack não chegou bem ao emissor (ficou corrompido). Neste caso despreza o pacote 1 porque já o recebeu anteriormente.
- Se estava a aguardar pelo pacote 1: semelhante ao que foi dito para o caso anterior.

[Slide 3.35]

Números de sequência apenas 0 e 1, porquê?
R: Porque é um protocolo stop-and-wait. Os pacotes são controlados um a 1 e não avança enquanto tudo não estiver ok para um pacote.

Quando comparado com a versão anterior, temos o dobro dos estados.

Como o receptor não sabe se o seu ACK/NACK ficou corrompido pelo caminho (não consegue inspecionar a sua viagem), este só se apercebe de que algo de errado aconteceu quando recebe pacotes em duplicado. Isto é, só assim o receptor toma conhecimento de que os ACK/NACKs não chegaram bem.

[Slide 3.36]

Podemos atingir o mesmo objectivo da versão 2.1 utilizando apenas ACKs: o ACK é enviado para o último pacote recebido correctamente (terá de ser incluido o número de sequencia no ACK). Assim, se o emissor receber dois ACKs iguais (com o mesmo número de sequencia) isso significa que o receptor não recebeu o último pacote bem (estava corrompido) e deverá retransmiti-lo.

[Slide 3.37]

Diferenças entre esta versão e a anterior:
- Receptor: terá de incluir agora o número de sequencia ao construir o pacote ACK (make_pkt()). Para o estado 'espera dados 0 de baixo': se recebeu dados corrompidos, ou com número de sequencia 1, tem de enviar novamente o ACK para o emissor porque o anterior não chegou bem. Algo de semelhante aconteceria para o estado 'espera dados 1 de baixo':
- Emissor: terá de verificar o número de sequencia do ACK recebido (isACK()). Se o emissor estava a aguardar o ACK 0 mas chegou o ACK 1, foi porque o ACK 0 não chegou bem ao receptor e deverá retransmiti-lo. O mesmo acontece se o ACK estiver corrompido.

[Slide 3.38]

Se existirem perdas o emissor poderia ficar eternamente à espera de um ACK que nunca mais chegava. Neste caso são necessários timeouts. Após um timeout o emissor deverá reenviar o pacote. Se o ACK estava apenas atrasado, e não perdido, serão recebidos acks em duplicado (com o mesmo número de sequencia) mas a versão anterior já contemplava isso: desprezava.

[Slide 3.39]

Após enviar um pacote de dados, o emissor acciona um timer (start_timer). Se houver timeout ou erros, os dados são reenviados e o timer começa novamente. Se dados ok, o timer pára.

Note-se que agora, enquanto espera por dados da camada superior, podem chegar os ACKs atrasados (evento). A acção a tomar é nenhuma porque já foram anteriormente reenviados. Isto é, enquanto aguardava por ACK pode ter havido retransmissões por timeout e, uma vez que o emissor passa para o estado de espera de dados assim que recebe um ACK (termina timer também), pode acontecer que cheguem ACKs (associados às retransmissoes) posteriormente.

[Slide 3.40]
Importante realçar que até agora usamos:
- checksums para detectar erros
- ACKs para que a fonte saiba se deve retransmitir
- números de sequencia para detectar duplicados
- timeouts para detectar perdas

[Slide 3.42]

Vamos assumir:
- Canal que liga emissor e receptor: 1 Gbps (bits por segundo)
- Tamanho de pacote: 1000 bytes (8000 bits)
- Atraso de propagação de um ponto a outro: 15 milisegundos (RTT=30 milisegundos)

Tempo necessário para colocar o pacote no link: L/R=8000/10^9=0,000008=8 microsegundos. O emissor terá então que aguardar que o pacote viaje até ao destino e que o ACK venha até ao emissor. Como o RTT é 30 milisegundos, a fração de tempo que o emissor está a transmitir é .008/30.008=0.00027 (para um máximo de 1). Isto é, apesar de a linha ser de 1Gbps, o throughput real é de 33KBps (ou 270Kbps em bits).

Conclusão: Os protocolos limitam a utilização dos recursos físicos!

[Slide 3.43]

Ilustração do que foi dito atrás.

No emissor:
- primeiro bit do pacote sai no instante t = 0.
- ultimo bit do pacote sai no instante t = L/R.
- ACK chega em t = RTT + L/R. Envia próximo pacote.

No receptor:
- primeiro bit do pacote chega em t = RTT/2.
- último bit do pacote chega em t = RTT/2 + L/R. ACK enviado.

Utilização: (L/R) / (RTT + L/R)

[Slide 3.44]

A solução para a baixa utilização é não funcionar com uma filosofia stop-and-wait, e permitir ao emissor enviar múltiplos pacotes sem ficar à espera de ACKs. Esta técnica é chamada de pipelining. Isto exige:
- Uma gama maior de números de sequencia; isto porque cada pacote em transito terá de ter um identificador único já que ainda não foi feito o seu acknowledgment.
- Buffering dos pacotes no emissor e receptor para reenvio se necessário. No emissor o buffer é usado para pacotes que não obtiveram o seu ACK. No receptor o buffer é usado para o caso de chegarem pacotes fora de ordem (pacote só são enviados para a camada acima se estiverem pela ordem correcta).
- A gama de números de sequencia necessária, e necessidades de buffering, depende da maneira como o protocolo responde a perdas, erros, e atrasos nos pacotes/acks. Existem duas abordagens: Go-Back-N e selective repeat.

Go-back-N = voltar atrás N
Selective repeat = repetição selectiva

[Slide 3.45]

Ilustração para três pacotes em trânsito.

No emissor:
- primeiro bit do 1º pacote sai no instante t = 0.
- ultimo bit do 1º pacote sai no instante t = L/R.
- ultimo bit do 3º pacote sai no instante t = 3 x L/R.
- ACK chega em t = RTT + L/R. Envia próximo pacote.

No receptor:
- primeiro bit do pacote chega em t = RTT/2.
- último bit do 1º pacote chega em t = RTT/2 + L/R. ACK enviado.
- último bit do 2º pacote chega em t = RTT/2 + (2 x L/R). ACK enviado.
- último bit do 3º pacote chega em t = RTT/2 + (3 x L/R). ACK enviado.

Utilização: (3 x (L/R)) / (RTT + L/R) = 0.0008 (triplicou comparado com o caso stop-and-wait)

[Slide 3.47]

GBN emissor:
- O emissor pode transmitir múltiplos pacotes/segmentos (N) sem ficar parado à espera dos acks.
- A imagem mostra a utilização de números de sequencia, do ponto de vista do emissor:
-- send_base: segmento mais antigo não confirmado.
-- nextseqnum: número de sequência mais pequeno não utilizado.
-- janela/window: gama de números de sequência, para pacotes transmitidos e não confirmados, sobre toda a gama de números de sequência possível. Por isso é que o protocolo GBN é chamado de protocolo de janela deslizante. Porquê limitar a janela ao tamanho N? Para que o emissor limite o seu envio de pacotes para o destino (flow control) e para adaptar o envio de pacotes ao congestionamento na rede (congestion control). Veremos mais tarde como é calculado.

O número de sequencia (que neste exemplo identifica pacotes) seria enviado no header do segmento. k bits permitem uma gama total de números de sequencia de [0,2^k-1]. Como os números de sequencia são limitados, toda a aritmética envolvendo números de sequência tem de usar o módulo de 2^k. Isto é, o espaço de números de sequência pode ser visto como um anel de tamanho 2^k onde o número de sequência (2^k)-1 é seguido do número de sequência 0.

Ainda estamos a falar apenas de princípios de transferencia fiável. Veremos adiante que no caso particular do TCP, são usados 32 bits para o número de sequência. Os números de sequência estão associados a bytes na stream de dados a enviar (não identificam pacotes mas sim bytes).

[Slide 3.48]

Não é necessário analisar este diagrama.

[Slide 3.49]

Não é necessário analisar o diagrama.

GBN receptor:
- Se foi recebido um segmento n sem erros em ordem (o número de sequência do segmento anterior era o n-1) então o receptor envia um ack referente ao segmento n, para o emissor, e entrega os dados à camada superior.
- Para os restantes casos o receptor despreza o segmento e reenvia um ack para o segmento mais recente recebido em ordem.
- Não faz buffering (mais simples). Apenas necessita de manter o expectedseqnum (variável que indica o número de sequência esperado) actualizado.

[Slide 3.50]

Exemplo de funcionamento do GBN:
- Está a ser usada uma janela de 4 segmentos
- O timeout mostrado está associado ao pkt2 uma vez que já foram recebidos o ack0 e ack1, que levaram à ativação de novo timer para o segmento mais antigo não confirmado.

[Slide 3.51]

- O GBN, quando tem uma janela grande, pode levar à retransmissão desnecessária de muitos segmentos (problema num segmento apenas pode levar à retransmissão de vários segmentos). Se a taxa de erros do canal aumenta existirão muitas retransmissões.
- No Selective Repeat o emissor apenas retransmite os segmentos com erros/perdidos, o que implica a confirmação individual, pelo receptor, de todos os pacotes recebidos. O emissor terá um timer para cada segmento não confirmado.

[Slide 3.52]

Na figura:
- No emissor, alguns segmentos já estão confirmados, outros não.
- No receptor, alguns segmentos são mantidos em buffer porque faltam outros segmentos que já deviam ter chegado. Os dados só são disponibilizados à camada superior quando estiverem pela ordem correcta.

[Slide 3.53]

Que eventos e acções existem no emissor:
- Se são recebidos dados da camada superior, o SR (selective repeat) emissor verifica o próximo número de sequência disponível. Se estiver dentro da janela, constrói o segmento e envia-o, caso contrário indica à camada superior que deverá tentar mais tarde ou então faz buffering.
- Para controlar perdas de segmentos é usado um timer para cada segmento.
- Quando recebe um ack, o emissor marca esse segmento como recebido. Se se tratava do segmento mais antigo (igual ao ponteiro send_base), o send_base avança para o próximo segmento ainda não confirmado. Se tinha feito buffering de algo que não cabia na janela, transmite agora.

Que eventos e acções existem no receptor:
- Se recebeu um pacote com número de sequência dentro da janela, envia ack mesmo para segmentos fora de ordem. Se está fora de ordem é colocado em buffer. Se está em ordem, disponibiliza à camada superior todos os

segmentos em ordem e avança o ponteiro rcvbase para o próximo segmento esperado.
- Se o pacote que recebeu já tinha sido confirmado, reenvia de novo o ack (o ack anterior tinha-se perdido).

Problema: O emissor e receptor não terão uma visão idêntica do que foi recebido correctamente ou não, e logo não terão janelas semelhantes. Isto pode trazer problemas uma vez que o conjunto de números de sequência é limitado. Vejamos a seguir um exemplo em que os números de sequência vão de 0 a 3 (4 no total) e a janela é de 3.

[Slide 3.54]

Exemplo sem problemas:
- os segmentos 3, 4 e 5 são armazenados em buffer no receptor. Estes são disponibilizados à camada superior em simultâneo com o segmento 2.

[Slide 3.55]

Exemplo com problemas:
- Suponhamos que os segmentos 0, 1 e 2 são recebidos correctamente. O receptor aguarda o quarto, quinto e sexto segmentos, com números de sequência 3, 0 e 1.
- Considere-se 2 cenários:
-- Os acks dos 3 segmentos perdem-se e o emissor retransmite os 3 pacotes.
-- Os acks dos 3 segmentos chegam correctamente, o emissor move a janela e envia os próximos segmentos com números de sequência 3, 0 e 1. Acontece que o novo pacote com número de sequência 3 perde-se mas o novo pacote com número de sequência 0 chega correctamente.
- Ambos estes cenários geram o mesmo estado de janela. Mas, no primeiro caso o receptor vai receber o primeiro segmento novamente (0) e pensa que é o quinto. Incorrectamente esse irá ser depois enviado para a camada superior. Qual o problema? Os números de sequência foram insuficientes para a janela definida, ou a janela é muito grande.
- Para SR: Tamanho da janela deve ser menor do que metade do tamanho do universo de números de sequência.

[Slide 3.56]

Vimos os principios de transferência fiável. Vamos ver agora como utiliza o TCP, o protocolo fiável usado pela Internet, esses principios.

[Slide 3.57]

O TCP é dito “orientado à conexão” porque antes de serem transmitidos os dados, os dois processos fazem handshake. Isto é, são enviados alguns segmentos preliminares para troca de parâmetros necessários à transferência dos dados. Muitas variáveis associadas à conexão TCP serão inicializadas.

Quando um processo cliente pretende comunicar com um processo servidor noutro host:

- O processo cliente informa a camada de transporte cliente que deseja estabelecer uma conexão com um processo servidor noutro host.
- A camada de transporte cliente estabelece uma conexão TCP com a camada de transporte no servidor (envia um segmento especial).
- O TCP servidor responde com um segundo segmento TCP especial.
- O TCP cliente envia um terceiro segmento especial, que também pode incluir dados.
Este procedimento é chamado de three-way handshake.

Os dados da camada de aplicação são colocados no TCP send buffer, um dos buffers criados na fase do handshake. O TCP irá buscar os dados a este buffer para criar segmentos (1 ou mais). A quantidade de dados máxima que pode ser colocada num segmento está limitada pelo MSS (maximum segment size). O MSS refere o número de bytes que podem estar no payload (dados) do segmento (isto é, não inclui header). Como é determinado o MSS? É visto qual o maior tamanho de frame que pode ser enviado pelo emissor (MTU, maximum transmission unit) e garante-se depois que o segmento cabe lá. Nas redes Ethernet MTU=1500, e MSS=1460 se não existirem opções nos headers da camada de transporte e rede. Note-se que a camada de ligação ainda acrescenta header, por isso o pacote que sai é maior que 1500 bytes (o tamanho irá depender da camada de ligação/placa que estiver a ser usada).

Nota: Links diferentes podem ter MTUs diferentes. Um pacote poderá atravessar links com diferentes MTUs. Se um router no caminho verifica que o pacote que recebeu é demasiado grande para o MTU do link de saida, faz fragmentação ao nível da camada de rede (apenas no IPv4). Ao permitir fragmentação, o protocolo IP possibilita que um datagrama seja dividido em pedaços mais pequenos que o datagrama original. O processo de fragmentação marca (header da camada de rede) os fragmentos para que a camada IP do destinatário possa montar os pacotes recebidos, reconstituindo o datagrama original. Veremos isto quando analisarmos a camada de rede.

[Slide 3.58]

- source and destination port: tal como o UDP.
- checksum: tal como o UDP.
- sequence e acknowledgment number: 32 bits cada. O "sequence number" numera o primeiro byte na seção de dados, e o "acknowledgment number" indica qual o próximo "sequence number" esperado (vindo do outro host).São usados pelo emissor e receptor na implementação de um serviço de transferência fiável. Referem-se a bytes e não pacotes.
- receive window: 16 bits. Usado para controlo de fluxo. Indica o número de bytes que o host está apto a receber.
- header length field: 4 bits, especifica o tamanho do header em words de 32 bits. O header pode ser de tamanho variável por causa das opções.
- flags: 6 bits:
-- ack: indica que o número no "acknowledgment number" é válido, isto é, o segmento em causa tem um acknowledgment para um segmento recebido com sucesso.
-- rst, syn, fin: são usados no estabelecimento e teardown da conexão como veremos adiante.

-- psh: indica que o receptor deve passar os dados para a camada superior imediatamente (não usado na prática).
-- urg: Indica que existem dados no segmento que a camada de aplicação no emissor marcou como urgentes. A localização do último byte destes dados urgentes é indicado em "urg data pointer". O TCP receptor deve informar a camada de aplicação quando isso acontece e enviar-lhe o ponteiro para os dados (não usado na prática).

[Slide 3.59]

- Número de sequência: número associado ao primeiro byte nos dados do segmento.
- Acknowledgement number: Sendo o TCP full-duplex o host A pode estar a enviar dados para o host B enquanto chegam dados do host B. O acknowledgment number que um host coloca num segmento é o número de sequência do próximo byte esperado.
- Exemplo do livro: ficheiro com 500 000 bytes, MSS=1000 bytes, primeiro byte dos dados com número 0. São construídos 500 segmentos de 1000 bytes de dados, o primeiro segmento tem sequence number 0, o segundo tem sequence number 1000, etc.

Nota: o TCP faz o acknowledgement de bytes até ao primeiro byte em falta. Logo, o TCP fornece "cumulative acknowledgment".

[Slide 3.60]

Exemplo no slide: Os números de sequencia iniciais para o cliente e servidor são trocadados no inicio (three-way handshaking). Note-se que os acknowledgments são enviados em segmentos contendo dados. Apesar de o último segmento não ter dados o segmento necessita de número de sequência.

[Slide 3.61]

O TCP usa timeouts quando existem perdas de segmentos.
Pergunta: Qual deve ser o tamanho do timeout?
Resposta: Deve ser maior que o RTT, que é variavel!
Pergunta: Como estimar o RTT?
Resposta: É medido o tempo entre a transmissão do segmento e a recepção do respectivo ACK (chamada amostra). É retirada uma amostra mais ou menos de RTT em RTT (valor antigo) e não para todo o segmento.

[Slide 3.62]

- O novo valor de EstimateRTT é uma média ponderada do EstimateRTT anterior e a amostra recente: alpha=0.125 e 1-alph=0.875 são os valores recomendados.
- Porquê o nome “exponential”? R: a influência das amostras decresce de forma exponencial (da amostra mais recente para a mais antiga).
Exponential weighted moving average (EWMA).

[Slide 3.63]

Calcular o valor de timeout usando apenas uma média ponderada não é suficiente. Devemos considerar também a variância. A expressão de DevRTT analisa a variação do RTT. É uma EWMA da diferença entre o SampleRTT e o EstimateRTT. Se têm pouca variação este será pequeno. Caso contrário será grande. O valor de timeout a usar inclui agora o EstimatedRTT e a margem 4*DevRTT. Esta margem é maior se os valores de sample RTT e EstimateRTT são muito variáveis. A decisão de 4 toma em consideração este e outros aspectos.
- beta=0.25, valor recomendado.

[Slide 3.65]

Vamos agora falar de como o TCP faz reliable data transfer:
- A gestão de um timer para cada segmento torna-se pesada. Por isso o TCP usa 1 timer apenas, que está associado ao segmento mais antigo ainda não confirmado.
- Vamos ver uma primeira versão em que ocorrem retransmissões apenas em caso de timeout; é depois vista uma versão mais complicada em que para além do timeout também são usados acks duplicados para sinalizar necessidade de retransmissão.

[Slide 3.66]

Note-se que:
- o valor para o intervalo de timeout usa o valor que vimos anteriormente: EstimatedRTT+4*DevRT.
- apenas é retransmitido o segmento mais antido não confirmado, e não todos.
- o TCP usa "cumulative acknowledgment".

[Slide 3.67]

Nota: No evento “ACK received, with ACK field value of y” a linha "sendbase=y" revela que se trata de um cumulative acknowledgment.

[Slide 3.68]

Vejamos 3 cenários:

Cenario 1: É enviado um segmento com número de sequencia 92 e 8 bytes de dados. O ack vindo de B tem o número de sequencia 100, mas perde-se. Quando o timout dispara o segmento é retransmitido. O host B vê que o número de sequencia do segmento que chegou refere-se a dados já confirmados, despreza os dados e envia um ack novamente.

Cenário 2: São enviados dois segmentos, e ambos os acks atrasam-se. Após o timeout é reenviado o primeiro segmento, e o timer é reinicializado. O segundo segmento não é reenviado. Como o ack do segundo segmento chega durante o novo timeout, o segundo segmento não é retransmitido.

[Slide 3.69]

Cenário 3: São enviados dois segmentos, e apenas o primeiro ack se perde. Como o segundo ack chegou antes de expirar o timeout, não são reenviados segmentos porque o host A sabe que até ao byte 120 chegou tudo bem.

O TCP tem a seguinte particularidade: O timeout duplica a sua duração para retransmissões consecutivas. O valor original é retomado nas restantes situações (transmissões normais). Isto porque se existiu timeout foi porque os routers provavelmente tinham as queues cheias. Ao insistir com muitas retransmissões pode-se estar a piorar a situação.

[Slide 3.70]

TCP receptor:
- Um dos problemas do timeout usado pelo TCP é que o timeout pode ter um período muito longo, provocando um delay muito elevado. Felizmente o emissor consegue detectar a perda de um segmento, antes de ocorrer o timeout, através dos 'duplicate acks'. 'duplicate ack' é um ack que reconfirma um segmento para o qual o emissor já recebeu um ack anteriormente.
- Porque gera o receptor um duplicate ack: quando chega um segmento fora de ordem, em vez de fazer o ack do segmento que chegou, o TCP envia um ack com o número de sequencia do próximo byte esperado. Logo, está a reconfirmar que recebeu corretamento o segmento cujos dados terminam no byte antes deste byte esperado. Quando os gaps forem preenchidos o TCP faz o ack do total, indicando qual o próximo byte esperado.

[Slide 3.71]

Como o emissor envia vários segmentos sem aguardar confirmação, o emissor pode receber vários "duplicate acks". Se o emissor receber 3 "duplicate acks" para os mesmos dados, então assume que o segmento se perdeu e retransmite. Porque não reage ao 1° duplicate ack? Porque o emissor não sabe se de facto é uma perda ou se se trata de um atraso.

[Slide 3.72]

Cenário de retransmissão rápida.

O TCP acaba por ser um híbrido do GoBackN (ex: cumulative ack) e SelectiveRepeat (ex: buffering).

[Slide 3.73]

Agora vamos ver como o TCP faz "flow control". Isto é, como consegue o TCP não enviar mais dados do que o receptor consegue receber. A parte de congestionamento da rede veremos adiante. É mais complicado.

[Slide 3.74-3.75]

O TCP consegue controlar o fluxo da seguinte forma:
- A parte receptora do TCP tem um buffer com tamanho: RcvBuffer.
- O processo na camada de aplicação lê do buffer de x em x tempo.

- A variável rwnd ("receive window", espaço em buffer não utilizado) é mantida actualizada e é usada para informar o emissor do espaço livre em buffer.
- O espaço livre em buffer é dado por:
rwnd=RcvBuffer-[LastByteRcvd-LastByteRead]
onde LastByteRcvd e LastByteRead são os números de sequencia do último byte recebido e do último byte lido respectivamente.
- O receptor mantém o emissor informado do espaço vazio no seu buffer (rwnd). Ver estrutura do header do segmento TCP. O emissor controla o envio de dados de acordo com este valor. Como?
-- mantem duas variáveis LastByteSent e LastByteAcked que indicam a quantidade de dados não confirmada.
-- O emissor garante que: LastByteSent-LastByteAcked <= rwnd

[Slide 3.76]

Antes de passarmos à parte de controlo de congestionamento, que é mais complicada, vejamos como são iniciadas/terminadas as conexões TCP.

[Slide 3.77-3.78]

Passos do handshake (convém ver estrutura do segmento TCP, slide 57):
1. O cliente envia um primeiro segmento especial TCP sem dados. Flag SYN=1. Este segmento é chamado segmento SYN. O cliente também envia um número de sequencia inicial (client_isn) que pode ser gerado aleatoriamente, ou simplesmente colocado a 0 (depende do sistema operativo).

2. O receptor ao receber este segmento terá que alocar buffer, variáveis. O receptor responde com um segmento especial TCP sem dados com flags: SYN=1 e ACK=1, acknowledgment field = client_isn+1, e o seu número de sequencia também gerado aleatoriamente ou colocado a 0 (server_isn). Este segmento é chamado segmento SYNACK.

3. O cliente ao receber o segmento SYNACK também aloca buffers e variáveis à conexão. O cliente envia um segmento para fazer o acknowledgement (coloca acknowledgment field = server_isn+1). Agora SYN=0, uma vez que a conexão já está estabelecida, e ACK=1. Este segmento pode conter dados e, nesse caso, o campo que contém número de sequencia será client_isn+1.

[Slide 3.79-3.80]

Para fechar a conexão:
- O cliente envia um segmento com o bit FIN=1.
- O servidor responde com um acknowledgement e fecha (não definitivo) a conexão. Depois envia um segmento com o bit FIN=1.
- O cliente recebe o segmento FIN e responde com um acknowledgement.

Primeiro envio contém: FIN=1, SeqNum=x;
Segundo envio contém: ACK=1, acknowledgement number=x+1;
Terceiro envio contém: FIN=1, SeqNum=y;
Quarto envio contém: ACK=1, acknowledgement number=y+1

Nota: Podem ser recebidos vários FINs se os ACKs se perderem ou atrasarem.
Nota: Ambos os lados não fecham imediatamente após o envio do FIN porque podem chegar segmentos atrasados.

Curiosidade: O nmap é um portscan de uso geral, que pode ser usado, para verificar as portas abertas num determinado host. O uso mais simples é escanear directamente uma máquina da rede:
nmap 192.168.0.3
Pode escanear uma porta específica:
nmap -sV -p 22543 192.168.0.4
Se ao fazer-se o scan de uma porta TCP (TCP SYN segment para porta especificada) é retornado um SYNACK é porque o destino tem essa porta aberta e não houve bloqueios pelo caminho. Se é retornado um RST do target host é porque este não tem essa porta aberta mas sabe-se que o segmento não foi bloqueado por nenhuma firewall. Se nada é recebido então o segmento SYN foi bloqueado.

[Slide 3.81]

As perdas podem ser o resultado de congestionamento na rede e ainda não vimos como este congestionamento pode ser tratado. Vamos primeiro ver alguns princípios e depois como o TCP os aplica.

[Slide 3.82]

Vamos de seguida analisar 3 cenários em que ocorre congestionamento na rede.

[Slide 3.83]

Cenario 1: Dois emissores e dois receptores, um router com buffer infinito (inrealista).
- Host A: envia dados a uma taxa lambda_in. Logo, o trafego oferecido ao router é lambda_in (considerando a situação mais simples em que não existem erros, retransmissoes, etc).
- Host B: Semelhante ao host A.
- Host A e B partilham um link de capacidade C e um router.
- A primeira figura mostra o throughput (lambda_out ou dados que chegam ao receptor) em função da taxa de envio (lambda_in). A segunda figura mostra o delay em função da taxa de envio. Para uma taxa de transmissão entre 0 e C/2 o throughput no receptor iguala o envio do emissor. Acima disto o throughput é sempre C/2. Este limite superior é resultado da partilha do link por 2 hosts a emitir à mesma taxa. Este limite nunca consegue ser ultrapassado daí o grafico do delay não exceder este valor no eixo do x.
- Podemos pensar que operar no limite da capacidade do link pode ser vantajoso porque estamos a aproveitar toda a largura de banda. Contudo operar no limite da capacidade do link provoca delays elevados.
- Podemos concluir que: o custo associado a uma rede congestionada é um queueing delay elevado quando a taxa de pacote se aproxima da capacidade do link.

Nota: A meu ver este slide é enganador porque se lambda_out acompanha lambda_in então não deveria haver atraso, para que isto seja possível. Vamos assumir que o grafico do throughput indica que é possível o throughput no receptor igualar o envio do emissor, mas isto será possível apenas para um tráfego "muito bem comportado" (sem rajadas, não real). O gráfico do atraso mostra uma situação mais real em que, tendo o tráfego algumas rajadas, ocorrem atrasos significativos à medida que lambda_in se aproxima de C/2 (porque existirão pacotes em queue).

[Slide 3.84]

Cenario 2: Dois emissores e dois receptores, um router com buffer finito.
- Consequência de buffer finito: drops de pacotes quando o buffer está cheio.
- Temos agora duas taxas de transmissão: a da camada de aplicação (lambda_in bytes/seg) e a da camada de transporte que inclui retransmissões (lambda_in' bytes/seg). Esta última é por vezes chamada de "offered load".
- O desempenho deste cenário irá depender de como são feitas as retransmissões. Consideramos a seguir 3 casos.

[Slide 3.85]

Caso a: O host A consegue, magicamente, descobrir se o buffer do router tem espaço livre ou não. Caso não realista. Neste caso não existem perdas e: lambda_in = lambda_in'= lambda_out

[Slide 3.86]

Caso b: O host A reenvia apenas os pacotes que de facto se perderam (e não os atrasados). Isto pode ser implementado usando timeouts muito elevados. Caso ainda não realista. Neste caso: lambda_in'> lambda_out. Uma parte dos dados não são originais mas retransmissões.

[Slide 3.88-3.89]

Caso c: O host A reenvia também pacotes que estão muito atrasados. Caso realista. lambda_in' consideravelmente maior que lambda_out. Uma parte mais significativa dos dados não são originais mas retransmissões.

Nota: "goodput" refere-se aos dados que de facto interessam (isto é, exclui retransmissões). "throughout" refere-se a dados enviados (isto é, inclui retransmissões)

[Slide 3.90]

Cenario 3: Temos quatro hosts com taxa lambda_in (igual para todos), múltiplos links no caminho, routers com buffer finito (todos com capacidade R).

[Slide 3.91]

Como existem vários routers no caminho, os pacotes que foram "dropped" num determinado ponto levam a que a capacidade de transmissão, até aí

usada, seja desperdiçada. No limite, à medida que lambda_in aumenta, o lambda_out pode tender para 0 (isto significa que o pacote apesar de passar um ou mais routers pode ser "dropped" noutro ponto da rede mais adiante).

[Slide 3.92]

O TCP, como não tem suporte de congestionamento da camada IP, faz com que cada emissor adapte a taxa de envio ao congestionamento detectado na rede. Isto coloca 3 questões:
- Q1: Como é que o emissor limita a taxa de envio?
- Q2: Como é que o TCP detecta o congestionamento no caminho entre si e o destino?
- Q3: Que algoritmo utiliza o emissor para adaptar o envio de segmentos ao congestionamento na rede?

Vamos responder a estas questões. Vamos assumir o envio de um ficheiro muito grande.

[Slide 3.93]

Para além da rwnd (receive window) de que falamos no controlo de fluxo, existe uma outra janela que é mantida no emissor: cwnd (congestion window)

Pergunta: Quanto deve ser reduzido na cwnd quando existe uma perda?
Resposta: O TCP reduz para metade após detectada uma perda (mas valor não pode ficar abaixo de 1 MSS). Isto é, multiplica por 1/2, daí usar-se o termo "multiplicative decrease".

Pergunta: Quanto deve ser aumentado na cwnd quando chegam acks?
Resposta: O TCP aumenta a cwnd com calma. Aumenta um pouco para cada ack que chega com o objectivo de aumentar 1 MSS a cada RTT. O aumento é aditivo, daí usar-se o termo "additive increase". Vamos ver que isto apenas é verdade quando são detectados 3 acks duplicados (fast retransmit).

O grafico mostra este efeito.

[Slide 3.94]

A expressão da taxa tem cwnd em bytes e RTT em segundos. Isto é, taxa=cwnd/RTT bytes por segundo. Temos de dividir pelo RTT porque o emissor só transmite mais segmentos depois de chegarem os acks dos segmentos em trânsito (que leva RTT a chegarem). A figura ao lado ilustra isto: primeiro são enviados cwnd bytes do emissor para o receptor e, depois passar o periodo de tempo RTT (contado desde que sai o primeiro segmento, incluido nos vários segmentos em cwnd bytes), começam a ser recebidos os acks no emissor.

Resposta às questões Q1, Q2 e Q3 do slide 3.96:
- Q1: Como é que o emissor limita a taxa de envio? Vimos no controlo de fluxo que associada a uma conexão TCP existem buffers e variáveis na parte emissora e receptora:

Receptor: rwnd=RcvBuffer-[LastByteRcvd-LastByteRead]
Emissor: LastByteSent-LastByteAcked <= rwnd

O mecanismo de controlo de congestionamento mantém a variável cwnd em ambos os lados da conexão. Agora:
Emissor: LastByteSent-LastByteAcked <= min{cwnd,rwnd}

Por simplicidade vamos assumir que o buffer no receptor é tão grande que podemos ignorar rwnd. Se não olharmos a detalhes, e assumindo que as perdas e atrasos são negligenciáveis, o emissor irá enviar cwnd bytes no inicio do RTT e no fim do RTT serão recebidos os respectivos acks. Assim, podemos dizer que a taxa de envio de bytes é taxa=cwnd/RTT bytes/segundo. Quando cwnd muda, a taxa também muda.

- Q2: Como detecta o TCP o congestionamento no caminho entre si e o destino?
Se existe congestionamento no router irá haver drops. O emissor detecta devido ao timeout ou as duplicate acks. Isto indica ao emissor que existe congestionamento no caminho até ao destino. Ao receber acks normalmente o emissor depreende que está tudo ok e aumenta a taxa.

- Q3: Que algoritmo utiliza o emissor para adaptar o envio de segmentos ao congestionamento na rede?
O algoritmo do TCP para controlo de congestionamento tem 3 componentes: 1) AIMD (additive-increase, multiplicative-decrease; 2) slow start e 3) reação a eventos timeout.

[Slide 3.95]

Note-se que no início da conexão o TCP coloca cwnd=1 MSS. Como pode existir muita largura de banda livre, seria ineficiente um aumento linear nesta fase inicial. Assim, adopta um crescimento exponencial. Vamos ver como ocorre esse crescimento exponencial.

Pergunta: Como é feito o crescimento exponencial?
Resposta: O valor de cwnd duplica a cada RTT. Isto é feito aumentando o cwnd de 1 MSS para cada ack que chega. Exemplo:
1º Envia 1 segmento, recebe 1 ack, cwnd=2
2º Envia 2 segmentos, recebe 2 acks, cwnd=4
3º Envia 4 segmentos, recebe 4 acks, cwnd=8
4º Envia 8 segmentos, recebe 8 acks, cwnd=16
…

[Slide 3.96 - 3.97]

Foi dito que no inicio da conexão existe um crescimento exponencial partir de 1 MSS (slow start) até ocorrer um evento de perda. Depois passa a um comportamento “com forma de dentes” (cwnd passa para metade e depois tem crescimento linear). Mas esta afirmação está incompleta:
a) Se o evento de perda foi detectado com 3 duplicate ACKs, o TCP comporta-se como descrito.
b) Se o evento de perda foi detectado com timeout, o TCP tem comportamento diferente: entra em slow start em que cwnd=1 MSS, e cresce

depois exponencialmente até atingir metade do valor que tinha anteriormente. Depois cresce linearmente.

Como é implementado? É definido um threshold (variável com valor limite) Porquê este comportamento? O gráfico do slide 101 mostra como agia uma versão antiga do TCP (TCP Tahoe) em que o comportamento b) é usado para ambos os tipos de evento. A nova versão (TCP Reno) cancela o slow start para o evento a). Assim aproveita melhor a largura de banda já que neste caso a rede mostra ser capaz de entregar alguma coisa. Isto é, se está a enviar acks é porque está a receber segmentos.

[Slide 4.1]
A camada de transporte, vista anteriormente, garante comunicação entre processos. Esta conta com a camada de rede (que vamos ver) para garantir a comunicação host-a-host.

Como a camada de rede incluir muitos conteúdos, foram criados dois capítulos que cobrem o plano de dados e o plano de controlo, respetivamente. Neste capítulo 4, iremos falar das funções do plano de dados.

[Slide 4.2]

NAT, IPv6 e DHCP foram retirados deste capítulo e ficam para redes II.

[Slide 4.3]

Tipos de comunicação 'cast':
- unicast – um para um
- broadcast – um para todos
- multicast – um para muitos (um nó fonte pode enviar uma cópia de um pacote a um subconjunto dos restantes nós da rede; será visto em redes II no contexto das redes multimédia)

[Slide 4.4]

O facto de os routers não terem camadas acima da camada de rede significa que não correm aplicações do utilizador e não implementam serviços de transporte como os descritos atrás. Todos os hosts e routers têm camada de rede (embora umas sejam mais minimalistas que outras).

A figura apenas mostra que os routers operam até à camada de rede e os hosts operam nas 5 camadas. Todas as camadas de rede (incluindo as dos hosts) garantem a entrega dos pacotes desde a origem até ao destino.

[Slide 4.5]

O papel da camada de rede é mover pacotes de um host emissor para um host receptor. Para isso necessita de duas funções:

- forwarding: quando chega um pacote é necessário mover o pacote do link/physical port de input para o link/physical port de output. Isto é feito com base em tabelas com determinadas informações. Acção local. Forwarding é uma função (a principal) implementada no plano de dados. Um

pacote pode ser bloqueado (e.g., destino proibido) ou duplicado (enviado para vários links de saída). Escala temporal na ordem dos nanosegundos.

- routing: será necessário determinar os caminhos para os pacotes (preencher as tabelas). Isto é feito por algoritmos de routing. Acção global ao nível de toda a subrede. Função implementada no plano de controlo. Escala temporal na ordem dos minutos.

Nota: Por exemplo o RIP é usado na camada 3, para encaminhamento, mas envia advertisements usando UDP (na camada 4).

[Slide 4.7]

As componentes de software, que correm os algoritmos de encaminhamento nos routers, trocam mensagens que contêm informação de encaminhamento (conforme o protocolo de encaminhamento que está a ser usado).

Nota: Podemos chamar "packet switches" aos switches e routers. Os switches são também chamados link-layer switches, e tomam as suas decisões de forwarding com base num campo da camada link-layer. Os routers tomam as suas decisões de forwarding com base num campo da camada de rede.

[Slide 4.8]

A abordagem anterior é a tradicional. Uma alternativa será haver um controlador remoto que calcula e distribui as tabelas de forwarding a serem usadas por cada router. Neste caso a o plano de controlo está fisicamente separado do router, e o router apenas possui o plano de dados.

O controlador remoto pode ser gerido pelo ISP ou por outra entidade. Como comunicam os routers e o controlador? Atraves de mensagens que incluem a info necessária. Esta é a abordagem usada pelas redes definidas por software (SDN), que veremos adiante.

[Slide 4.10]

Muito genericamente um router terá: ports de input e output, uma forma de ligar os ports de entrada aos ports de saida (existem várias arquitecturas para esta ligação), e um processador de routing que executa os protocolos de routing, mantém tabelas de forwading e informações de routing.

[Slide 4.11]

Os input ports têm funções:
- da camada física: faz a terminação do link de entrada (função de transmissão/recepção de bits).
- da camada de ligação (data-link): necessário para comunicação com a camada de ligação no outro extremo. Esta camada preocupa-se com a transferencia de dados entre nós adjacentes. Esta função e a anterior são as funções incluidas numa placa de rede.

- forwarding, queueing: muitos routers mantêm uma cópia da tabela de forwarding nos input ports por questões de rapidez. Assim, os input ports verificam qual o port de saida para o pacote e colocam o pacote no switch fabric de forma a este emergir no port de saida certo. Pode haver buffering se a consulta da tabela de forwarding e o envio para o switch fabric fôr lento quando comparado com a chegada dos pacotes.

Nota: quando um PC faz de router o routing processor é o CPU e os input e output ports são as interfaces. A comutação é feita pelo CPU: o port de input gera uma interrupção ao processador, o pacote é copiado do input port para a memória, o processador verifica o port de saida na tabela de forwarding e copia o pacote para o buffer do port de output.

[Slide 4.12]
Para já vamos assumir forwarding com base no IP de destino. Mais adiante veremos outros critérios para forwarding.

[Slide 4.13]

Para um endereço de 32 bits existem 4 biliões de entradas possíveis. Vamos assumir que um router aceita a seguinte informação, mostrada no slide, relativa a cada interface de saida (0-3).

[Slide 4.14]

Neste caso não seriam necessários as 4 biliões de entradas na tabela. Bastava colocar also parecido ao que mostra o slide. O Prefix Match termina no digito a partir do qual todas as combinações de bits são aceites para essa entrada. Isto é, o router faz o match do prefixo com o endereço de destino que vem no pacote e o que coincide é o link de saida.

Exemplo 1: Coincide com a primeira entrada
Exemplo 2: Coincide com a segunda e terceira entradas. Neste caso o router optará pelo "match" mais longo (mais específico), isto é, segunda entrada.
Se não existe match envia para a última entrada.

Problema actual: existem cada vez mais gaps nas gamas de IPs referentes a um bloco de números. Isto não permite que existam poucas entradas e, por isso, as tabelas de routing estão cada vez maiores.

[Slide 4.16]

O output port transmite os pacotes, que estão em buffer, pelo link de saida. A politica de scheduling (FIFO, Weighted queuing, etc) permite introduzir algum QoS.

[Slide 4.17]

Neste exemplo assume-se que todas as linhas têm a mesma capacidade de transmissão, e que o switch opera a 3x a capacidade das linhas. Três pacotes foram transferidos para o buffer de output e aguardam transmissão. No instante seguinte um pacote foi transmitido e um outro

pacote é colocado no buffer. Em caso de excesso o pacote é dropped.
Existem várias politicas de drop (escolha do pacote a ser feito o drop).