Redes

Notas 2

As redes de telecomunicações dividem-se em:

- Redes comutadas por circuitos:
  - Multiplexagem por divisão de frequência (FDM)
  - Multiplexagem por divisão de tempo (TDM)
- Redes comutadas por pacotes:
  - Datagramas
  - Circuitos virtuais (VCs)

Redes de datagramas:

- Rede onde o endereço de destino dum pacote determina o próximo nó.
- Pacotes para o mesmo destino podem ter diferentes rotas.

Redes de circuitos virtuais

- Rede onde cada pacote leva um identificador de circuito que determina o próximo nó.
- Caminho fixo determinado no estabelecimento da ligação.
- São necessárias tabelas para manter o estado e informação dos circuitos virtuais.
- Pode permitir dar algumas garantias de qualidade de serviço.

Núcleo – Consiste numa malha de nós de encaminhamento (routers) que estão ligados entre si.

Abordagens na construção do núcleo de uma rede:

- ***Comutação de circuitos***
  - É dedicado um circuito real por ligação.
  - Os recursos necessários num caminho são reservados durante a sessão de comunicação.
  - Quando é estabelecido um circuito é reservada uma taxa de transmissão constante nos links utilizados durante a sessão de comunicação.
  - A capacidade de um link depende da capacidade de comutação dos extremos.
  - Se um link suporta n ligações então uma ligação entre dois hosts obtem 1/n da largura de banda do link.
  - Um link pode possuir varias conexões, pois a sua largura de banda está dividida em partes que são atribuídas a conexões.
- ***Comutação de pacotes***
  - Os recursos não são reservados à priori.
  - Como não é reservada largura de banda, se a rede estiver congestionada os pacotes têm de aguardar em queue.
  - Fornece um serviço **best-effort** dado que vários utilizadores podem usar a largura de banda disponível.
  - Normalmente usa transmissão **store-and-forward** em que o comutador tem de receber todo o pacote antes de o poder transmitir para outro link.
  - É introduzido um atraso store-and-forward proporcional ao tamanho do pacote.
  - Não existe nem divisão da largura de banda em partes, nem recursos dedicados.

Nem todas as redes podem ser classificadas em redes de circuitos ou de pacotes puras.

A internet usa comutação de pacotes.

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.

Assumindo que o atraso nas queues, de propagação e de processamento são negligenciáveis.

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

Nas **Redes Ethernet** os pacotes têm tamanho mínimo de 64 bytes e máximo de pouco mais que 1500 bytes

Existe congestionamento quando:

- Existe uma ligação ao router mas a sua velocidade de processamento impede que pacotes saiam à mesma velocidade que entram.
- Existem várias ligações ao router que, em conjunto, excedem a capacidade do link de saída.

Multiplexagem estatística - Quem transmite mais tem mais probabilidade de ter dados à saída.

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

Time Division Multiplexing (TDM)

- O tempo é dividido em frames de duração fixa. Cada frame é dividido 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.

Comparação de comutação de pacotes e comutação de circuitos:

- **Pacotes**
  - Não indicado para aplicações de tempo real devido aos atrasos imprevisíveis (sobretudo de queues).
  - Permite maior utilização da largura de banda
  - 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%.
- Na comutação de circuitos é necessário reservar um circuito de 100kbps para cada utilizador. Assim, para um link de 1Mbps, 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, a taxa de chegada de dados ao link é igual ou menor à taxa máxima de saída do link (1Mbps).

A **comutação de pacotes** é boa para tráfego de **rajada** (tráfego imprevisível), porém existem **atrasos** e **perdas de pacotes** quando existe congestionamento.

A **rede de acesso** situada na periferia da internet, que liga os end systems acedem à internet, está ligada à Internet através de uma **hierarquia de ISPs:**

- **Access ISPs**
  - ISPs residenciais, universidades, empresas
  - Mais abaixo nesta hierarquia
- **Tier-1 ISPs:**
  - Topo da hierarquia
  - Existem poucos.
  - É 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.
  - Estes ISPs estão ligados entre si, e a tier-2 ISPs
  - Têm cobertura internacional.
- **Tier-2 ISPs**
  - Têm uma cobertura regional
  - São clientes Tier-1 ISPs.

POP (Point Of Presence) - Central com equipamento de um ISP ('nó'), que liga os clientes ao ISP Tier-1.

IXP (Internet Exchange Point) - Localização onde vários ISPs estão presentes, se ligam e partilham meios de comunicação. Isto reduz custos e melhora o desempenho.

Quando a taxa de chegada excede a capacidade do link de saída, os pacotes são armazenados em buffers/queues e aguardam pela sua vez de sair.

Fontes de atraso:

- **Processamento no nó –** Verifica se existem erros e determina o link de saída.
- **Atraso nas queues –** Porque ficam a aguardar a sua vez de serem transmitido. Depende do congestionamento no router.
- **Atraso de transmissão –** L/R
- **Atraso de propagação –** Tempo que um bit leva a propagar desde o início até ao fim do link. O atraso de propagação é igual a tamanho do link/velocidade de propagacao no meio.

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.

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 10° terá algum atraso.

La/R 1

- Intensidade de trafego.
- Importante no dimensionamento das queues
- a = media de chegada de pacotes
- La/R > 1 – A taxa de chegada de bits excede o que é despachado e a queue vai crescendo.
- La/R < 1
  - 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 bursts 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.

Trace-route:

- Um pequeno programa em que o utilizador especifica um hostname/IP e são enviados vários pacotes especiais em direcao ao destino, com o objetivo de descobrir os atrasos da rede.
- 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, que possuem TTL=0, e envia a mensagem ICMP Time_Exceeded.
- O tracerout retorna varias linhas com o número do router no caminho, o seu nome e o IP, e no fim os 3 atrasos obtidos.

Throughput Medio = bits / tempo que demorou a transferir.

Aplicações como telefone e real-time video: querem baixo atraso e throughput consistente.

Aplicações como transferencia de ficheiros: atraso não é crítico.

Exemplo: Está a ser feito um download de um ficheiro 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}.

Exemplos de protocolos por camada:

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

Mensagem - É um pacote da camada de aplicação.
Segmento – É um pacote da camada de transporte (possui o header da camada de transporte + mensagem da camada de aplicação).
Datagrama - É um pacote da camada de rede (possui o header da camada de rede e do segmento da camada de transporte)
Frame - É um pacote da camada de ligação (possui o header da camada de ligação e do datagrama da camada de rede)

Percurso de uma mensagem:

- 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.

Programas que correm em dispositivos, como routers ou switches, são da responsabilidade do fabricante.

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.

Arquitetura cliente-servidor:

- Existe um host servidor, que está sempre ligado e com um IP permanente, que serve os pedidos vindos dos hosts clientes.
- Os clientes não comunicam entre si directamente.
- Quando o número de clientes é elevado, podem ser usados clusters (chamados "server farm") para criar um server virtual potente.

Peers:

- Pares de hosts
- Comunicam entre si directamente sem passar por um servidor.
- Os peers comportam-se como servidor e como cliente.
- Ex: Torrent

Os processos que trocam mensagens através da rede enviam/recebem mensagens através do seu socket.

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.
- Interface entre a camada de aplicação e a camada de transporte
- Um socket não é uma conexão, é sim o extremo de uma conexão.

Port number

- Identificador que define que serviço/aplicação
- Tem 16 bits: inteiro de 0 a 65535
  - 0-1023 – Well-known ports (serviços/aplicações essenciais, como mail, dns, etc)
  - 1024 a 49151 – Atribuidos e controlados pelo IANA
  - 49152 a 65535 – Dynamic ports (de atribuição dinamica pois aplicações como os nossos browsers nao estão sempre ativos)

Notas 3

Response codes:

- 200 OK – Request succeeded
- 301 Moved Permanently – Requested object moved (new location specified later in this msg)
- 400 Bad Request – Request msg not understood by server
- 404 Not Found – Requested document not found on this server
- 505 HTTP Version Not Supported

Os **cookies** são utilizados quando 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.

Quando um pc recebe uma resposta HTTP ela vai incluir um header com o numero de identificação do cookie "Set-cookie: num". Agora, quando o pc entrar num site do mesmo servidor ele vai saber quem é.

Problemas dos cookies:

- Cada cookie ficará associado um computador, conta de utilizador e web browser. Se uma pessoa usar múltiplas contas, computadores e browsers tem vários cookies, não identificando corretamente.
- Os cookies são trocados entre o servidor e o browser, por isso, estão visíveis a todos utilizadores que tenham um sniffer (ex: wireshark) na rede.
- **Cookie poisoning** – Quando um atacante modifica o conteudo de um cookie. Se um cookie contém o valor a pagar, por exemplo, o atacante pode lá colocar um valor mais baixo. Exemplo:
  GET /store/buy.asp?checkout=yes HTTP/1.0
  Cookie: SESSIONID=570321ASDD23SA2321; BasketSize=2; Item1=2892; Item2=3210; TotalPrice=7500;

Proxy Server – Guarda uma cópia dos objectos solicitados recentemente.

Como funciona um proxy server?

- 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 objecto solicitado envia-o ao browser.
- Se não tiver o objecto, o proxy server abre uma conexão TCP para o servidor original e envia um HTTP request solicitando o objecto.
- O servidor original envia o objecto num HTTP response ao proxy server.
- Este armazena o objecto no seu disco e envia uma cópia ao browser cliente usando um HTTP response.

Exemplo:

- O router da rede institucional está ligado ao router na Internet por um link a 1.5 Mbps;
- O tamanho médio de um objecto é 100K 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*100K/1000000000 = 0.0015.= 0.15%. Isto é, 15 pedidos vezes o tamanho médio de um objecto a dividir por 1 Gbps que é uma característica da LAN institucional.
- Utilização do access link: 15*100K/1500K= 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.

Esta solução baixa a intensidade de tráfego no link de acesso para 15*100K/150000K= 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.

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.

Caching reduz o tempo de atraso, porém introduz um novo problema: O objecto em cache pode ficar desactualizado e ser diferente do objecto no server.

Conditional GET:

- Mecanismo HTTP que permite que o cache verifique se os objetos estão desatualizados.
- Um request é conditional get quando usa o método GET e inclui o header “If-Modified-Since”.

Notas 4

Componentes do sistema de email:

- User Agent / Mail Readers – Permite aos utilizadores ler, responder, enviar emails.
- Servidores de email – São o core da infra-estrutura de email. É responsável pela autenticação do utilizador no acto de leitura do email, e por resolver falhas.
- **SMTP**
  - Protocolo principal 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.
  - 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.

SMTP faz conexão TCP é direta entre servidores (sem intermedios).

Quando o mail server quer entregar o email a outro server, tem de resolver um nome bob@empresa.com. Terá que ser feito um query (**tipo MX**) ao servidor de DNS do domínio empresa.com, que responderá com o IP.

Como são transferidas as mensagens usando o SMTP?

- Primeiro o cliente SMTP (servidor de mail que envia) estabelece uma conexão TCP na porta 25 do servidor SMTP (servidor de mail que recebe).
- Depois ocorre um handshaking. Nesta fase o SMTP cliente indica qual o email de origem e o email do destino.
- Depois o SMTP cliente envia a mensagem
- Se tiver outra mensagem para enviar o SMTP cliente repete o processo; caso contrário fecha a conexão TCP (conexão persistente).

As linhas com "C:" referem-se às linhas enviadas pelo cliente, para o seu TCP socket
As linhas com "S:" referem-se às linhas enviadas pelo servidor para o seu TCP socket.
A linha com "." indica fim da mensagem.

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

Difrenças entre SMTP e HTTP:

- HTTP é um pull protocol, ou seja, carrega informação.
- SMTP é um push protocol, ou seja, envia informação.
- Em HTTP a conexão TCP é iniciada pela máquina que quer receber o ficheiro.
- Em SMTP a conexão TCP é iniciada pela máquina que quer enviar o ficheiro.
- Usam codificações diferentes.

Em SMTP os dados são codificados da seguinte forma:

- 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.

POP3:

- Protocolo simples e limitado.
- O modo **Download and Delete** do POP3 tem o problema que não se podem ver os emails em maquinas diferentes. Por outro lado o modo **Download And Keep** mantem uma copia no servidor.
- Permite o utilizador pode mover emails para folders após puxar o email.

Fases do POP3:

- O user agent do cliente estabelece conexão TCP na porta 110
- Ocorre a 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.
- Fase actualização - Ocorre quando é feito o quit. Apaga mensagens marcadas e faz o log off do servidor de email.

IMAP:

- Permite que o utilizador crie folders e mova mensagens entre folders.
- Permite que se associe emails a pastas.
- Permite recolher apenas alguns objectos dos emails. Isto é importante quando a largura de banda é pequena e queremos apenas ver, por exemplo, o sender ou subject sem os anexos.

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.
- Normalmente usa-se IMAP para ser dada a possibilidade de criar folders e organizar msgs em folders.

DNS:

- Uma base de dados distribuida implementada numa hierarquia de servidores de DNS
- Usado para traduzir um hostname em endereço IP.
- Pode adicionar algum atraso, quando o endereço IP não está muitas vezes na cache de um servidor de DNS por perto.
- 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.

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 poderiam haver grandes atrasos.
- Teria uma base de dados enorme e actualizado com demasiada frequência.

Classes de servidores de DNS:

- Root servers - Têm informação sobre os IPs dos TLD servers.
- 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 seu servidor authoritative ou pode pagar a um fornecedor de serviços para armazenar os seus registos num authoritative DNS server do fornecedor.

Servidores locais ou "**default name servers**" agem como proxy fazendo o forward do query para a hierarquia de servidores de DNS.

Normalmente o query do host para o local server é recursivo e os outros não.

O **DNS** usa o **caching** de forma a diminuir os atrasos e o número de mensagens DNS a fazer ricochete na Internet.

Servidores DNS descartam a informação em cache normalmente após 2 dias.

Um **servidor local de DNS** também armazena em cache os números IP de servidores TLD para que possa fazer o bypass dos root servers. Os root servers estão armazenados num ficheiro.

Tipos de RR's:

- **Type=A** – Retorna o IP do host
  - name = hostname
  - value = Ip
- **Type=MX –** Retorna o servidor de email
  - Name = email
  - Value = Servidor Email
- **Type=NS –** Retorna o hostname do authoritative server
  - Name = domínio (ex: foo.org)
  - Value = Hostname do authoritative server
- **Type=CNAME –** Transforma um alias num nome real (canonical name)
  - Name = alias
  - Value = Canonical name

Mensagens de DNS podem ser: query ou reply.

Formato das mensagens DNS:

- **Identificação** – Numero de 2 bytes que identifica o query. É copiado para o reply para se associar o query com o reply.
- **Flags** – Indicam:
  - Se é um query ou um reply
  - Se o DNS server é authoritative
  - Se a query deve ser recursiva
  - Se o DNS server suporta recursividade (se for reply)
- 4 campos de "**number of**", que indica o numero de ocorrências nas 4 secçoes seguintes
- **Questions** - Inclui o nome e tipo (A, PTR, MX, ...) do que está a ser questionado.
- **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** - Authority records (do tipo NS) nesta seção. Indicam nomes de servidores authoritative.
- **Additional Info** - Contém RRs que estão relacionados com o query, mas não são resposta à questão.

Sequencia de passos duma query DNS:

- O teu 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 nosso servidor local de DNS contendo estes dois RRs.
- O servidor local de DNS envia um query para o ip recebido (?) a pedir um registo do tipo A para www.networkutopia.com (site que procuramos).
- Ao receber a resposta, o servidor local envia-a para o nosso host.

Notas 5

Camada de transporte:

- Garante a comunicação logica entre processos de aplicações que correm em hosts diferentes.
- Fornece à camada superior um canal 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. O que irá dificultar a tarefa deste canal é o facto de na camada de rede o canal não ser fiável (unreliable). O protocolo IP é então um **unreliable data transfer (udt) protocol**.

Comunicação lógica:

- Do ponto de vista das aplicações é como se estas estivessem directamente ligadas.
- 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.

Segmentos:

- Os pacotes da **camada de transporte**
- 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 (**datagramas**). No receptor a camada de rede extrai os segmentos, que estão dentro de datagramas, e envia-os para a camada de transporte.

A **camada de rede** fornece uma comunicação logica entre hosts.

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 cartas. As cartas são entregues pelos serviços de correios (equivalente **ao protocolo IP**). Acontece que em cada casa existe uma criança que recolhe as cartas (dos que vivem na mesma casa) e entrega-as ao carteiro, e recebe as cartas entregues pelo mesmo, distribuindo-as pelas crianças da casa (equivalente ao **protocolo de transporte**).

O **protocolo IP** fornece serviço **best-effort**, ou seja, o IP faz um esforço para entregar os segmentos entre os hosts mas não dá garantias.

Serviços best-effort não garantem:

- A integridade dos segmentos
- A entrega de segmentos (se queues congestionadas estes podem ser desprezados)
- A entrega dos segmentos pela sua ordem pois podem 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.

TCP inclui:

- Controlo de fluxo - Ajuste à velocidade do receptor
- Controlo de congestionamento - Evitar congestionar uma rede que já está congestionada.

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

Multiplexagem na camada de transporte:

- São recolhidos dados, de diferentes sockets, e estes são partidos em pedaços.
- A cada pedaço é adicionada informação (header) necessária na recepção.
- Os segmentos são enviados para a camada de rede.

Demultiplexagem na camada de transporte:

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

O port origem e o port destino são colocados no header da camada de transporte. A camada de rede coloca o endereço IP origem e destino no seu header.

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

Um 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.

UDP apenas multiplexa/desmultiplexa e verifica erros (perante um erro faz drop).

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

Tamanho do **header** do **UDP** são 8 bytes, enquanto o **TCP** tem 20 bytes.

Aplicações que usam UDP:

- **DNS** - Muitas aplicações (ex: web, mail, etc) fazem queries ao DNS para resolver nomes, por isso terá que ser uma operação rápida.
- **RIP** - Protocolo para actualização de tabelas de routing.
- **SNMP** - Protocolo usado para monitorar e controlar serviços e dispositivos de uma rede TCP/IP.

CheckSum:

- Valor que indica o numero de bits em transmissão
- Usado para detector erros na transmissao de dados.

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 para passar os dados ao canal reliable.
- **rdt_rcv()** - Usado pela camada de baixo para passar os dados ao canal reliable. 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.

Rdt 1.0:
- Canal sem erros e sem perdas.
- Separa a FSM do sender e do receiver.
- O sender envia pelo canal e o receiver lê do canal.

Rdt 2.0:
- Ainda não são consideradas perdas.
- Possui um mecanismo para deteção de erros: **checksum**
- Feedback do receptor – **ACKS** E **NAKS.**
- Retransmissão de pacote caso seja recebido um NAK (negative acknowledgement) pelo emissor.

Rdt 2.1:
- Resolve o problema do Rdt 2.0: não controla a existência de erros nos ACKs e NAKs. O emissor não sabe verdadeiramente o que aconteceu no destino.
- 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.
- Possui o dobro dos estados da versão anterior
- O receptor só toma conhecimento de que os ACK/NACKs não chegaram bem, quando recebe pacotes duplicados

Diferenças entre Rdt 2.0 e Rdt 3.0:
- **Receptor:**
  - Terá de incluir o nº de sequencia no ACK.
  - Se recebeu dados corrompidos, ou com número de sequencia errado, tem de enviar novamente o ACK para o emissor porque o anterior não chegou bem.
- **Emissor:**
  - Terá de verificar o número de sequencia do ACK recebido. 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.

Notas 6

Timeouts:
- Se existirem perdas o emissor poderia ficar eternamente à espera de um ACK que nunca mais chegava.
- 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.
- Após enviar um pacote, o emissor aciona um timer. Se os dados chegarem, o timer pára.
- Deve ser maior que o RTT.

Exercicio:

- 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)
- **Resolução:**
  - 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).

Um exemplo qualquer:

- O primeiro bit do pacote sai no instante t = 0.
- O ultimo bit do pacote sai no instante t = L/R.
- O ACK chega em t = RTT + L/R.
- É enviado o próximo pacote.
- O primeiro bit do pacote é recebido a t = RTT/2.
- O ultimo bit do pacote é recebido em t = RTT/2 + L/R.
- O ACK enviado.

Stop And Wait:

- O emissor envia um pacote, e espera o seu ACK para puder enviar outro pacote

Pipelining:

- O emissor envia vários pacotes sem esperar pelos seus ACKS.
- Possui mais números de sequencia - 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.
- Os dois pontos anteriores dependem 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 (Sliding Window) = Reenviam-se os últimos N pacotes.
Selective repeat = Reenviam-se apenas os pacotes selecionados.

Emissor Go-Back-N:

- O emissor transmite múltiplos pacotes(N) sem ficar esperar pelos acks.
- Congestion Window: gama de números de sequência, para pacotes transmitidos e não confirmados, sobre toda a gama de números de sequência possível.
  - O tamanho da janela é limitado 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**).

Receptor Go-Back-N:

- Se foi recebido um segmento n sem erros em ordem então o receptor envia um ack referente ao segmento n.
- Caso existam problemas, o receptor reenvia um ACK do segmento mais recente recebido em ordem.

- Não faz buffering, apenas necessita de manter o numero de sequencia esperado actualizado.

Selective Repeat:

- Ao enviar um segmento o 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.
- Se o receptor 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.
- Se o pacote que recebeu já tinha sido confirmado, reenvia de novo o ack (o ack anterior tinha-se perdido).
- O tamanho da janela deve ser menor do que metade do tamanho do universo de números de sequência.

GBN vs Selective Repeat:

- O **GBN**, quando tem uma janela grande, pode levar à retransmissão desnecessária de muitos 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.

Problema do Selective Repeat:

- 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.
- **Exemplo:**
  - 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.
  - 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.

O **TCP** é dito "**orientado à conexão**" porque antes de serem transmitidos os dados, os dois processos fazem handshake.

Three-Way Handshake:

- A camada de transporte cliente estabelece uma conexão TCP com a camada de transporte no servidor (envia um segmento especial).
- O TCP server responde com um segundo segmento TCP especial.
- O TCP cliente envia um terceiro segmento especial, que também pode incluir dados.

MSS (maximum segment size) - Refere o número máximo de bytes que podem estar no payload (dados) do segmento (isto é, não inclui header).

O **MSS** é calculado pela diferença entre o **MTU** (Maximum Transmission Unit) e o tamanho dos headers.

Nas redes Ethernet MTU=1500, e MSS=1460 se não existirem opções nos headers da camada de transporte e rede.

Links diferentes podem ter MTUs diferentes, logo 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).

Estrutura de um segmento TCP:

- Source port – 2 Bytes
- Destination port – 2 Bytes
- Sequence Number - Numera o primeiro byte na seção de dados (4 bytes)
- Acknowledgment number - Indica qual o próximo "sequence number" esperado (4 bytes)
- Flags – 2 bytes.
- Receive Window: 2 bytes. Usado para controlo de fluxo. Indica o número de bytes que o host está apto a receber.
- CheckSum – 2 bytes.
- Urg Data Pointer – 2 bytes
- Options – Tamanho variavel
- Application data – Tamanho Variavel

O **Sequence Number** aumenta de acordo com o tamanho do MSS. Por exemplo, para vários segmentos com MSS=1000 bytes, o primeiro terá S.N.= 0 e o segundo S.N.= 1000.

Cumulative Acknowledgement:

- Usado pelo TCP
- Cada ACK diz que todos os pacotes chegaram até aquele numero de sequencia.

Notas 7

TCP usa apenas **1 timer**, que está associado ao segmento mais antigo ainda não confirmado.

O valor para o **intervalo de timeout** usa o valor que vimos anteriormente: EstimatedRTT+4*DevRT.

Em TCP apenas é retransmitido o segmento mais antigo não confirmado, e não todos.

Exemplos do funcionamento do TCP:

- **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.
- **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 transmissões.
- 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.

Um dos **problemas do timeout usado pelo TCP** é que 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'** (um ack que reconfirma um segmento já confirmado).

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.
- 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 1º duplicate ack? Porque o emissor não sabe se de facto é uma perda ou se se trata de um atraso.

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

Flow Control - Como consegue o TCP não enviar mais dados do que o receptor consegue receber.

Flow Control do TCP:

- A parte receptora do TCP tem um buffer.
- O processo na camada de aplicação lê do buffer de x em x tempo.
- A variável rwnd (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.

Passos do handshake:

- O cliente envia um segmento TCP com a flag SYN=1 (segmento SYN), com um número de sequencia inicial que pode ser gerado aleatoriamente, ou simplesmente colocado a 0 (depende do sistema operativo).
- O receptor ao receber este segmento aloca o buffer e variáveis, e responde com um segmento especial TCP sem dados com flags: SYN=1 e ACK=1, e o seu número de sequencia também gerado aleatoriamente ou colocado a 0 (segmento SYNACK).
- O cliente ao receber o segmento SYNACK aloca buffers e variáveis à conexão. O cliente envia um segmento para fazer o acknowledgement. Agora SYN=0, uma vez que a conexão já está estabelecida, e ACK=1. Este segmento pode conter dados.

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.

Passos para fechar a conexao:

- 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

Podem ser recebidos vários FINs se os ACKs se perderem ou atrasarem.

Cenários de Congestionamento:

- **Cenário 1: Dois emissores e dois receptores, um router com buffer infinito (inrealista) (Slide 3.83)**
  - Host A: envia dados a uma taxa lambda_in. Logo, o trafego oferecido ao router é lambda_in (considerando que não existem erros, retransmissoes, etc).
  - Host A e B partilham um link de capacidade C e um router.
  - 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.
  - Operar no limite da capacidade do link provoca delays elevados.
  - O custo associado a uma rede congestionada é um queueing delay elevado quando a taxa de pacote se aproxima da capacidade do link.
- **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).
    - O desempenho deste cenário irá depender de como são feitas as retransmissões.
        - **Caso a:** O host A consegue, magicamente, descobrir se o buffer do router tem espaço livre ou não (não realista). Neste caso não existem perdas e: lambda_in = lambda_in'= lambda_out
        - **Caso b:** O host A reenvia apenas os pacotes que de facto se perderam (e não os atrasados). Isto pode ser implementado com timeouts muito elevados (não realista). Neste caso: lambda_in'> lambda_out. Uma parte dos dados não são originais mas retransmissões.
        - **Caso c:** O host A reenvia também pacotes que estão muito atrasados (realista). lambda_in' muito maior que lambda_out. Uma parte mais significativa dos dados são retransmissões.
- **Cenario 3: Temos quatro hosts com taxa lambda_in (igual para todos), múltiplos links no caminho, routers com buffer finito (todos com capacidade R).**
    - 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.

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.

O emissor mantem a rwnd (receive window) e a cwnd (congestion window).

Quanto deve ser reduzido na cwnd quando existe uma perda?
O TCP reduz para metade após detectar uma perda (mas valor não pode ficar abaixo de 1 MSS).

Quanto deve ser aumentado na cwnd quando chegam acks?
Aumenta-se 1 MSS a cada RTT até ocorrer uma loss. Isto apenas é verdade quando são detectados 3 acks duplicados (fast retransmit).

Taxa de Envio = cwnd/RTT bytes por segundo.

O **mecanismo de controlo de congestionamento** mantém a variável cwnd em ambos os lados da conexão.

Como é que o emissor limita a taxa de envio?
- Assumindo que o buffer no receptor é tão grande que podemos ignorar rwnd, e 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.

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 aos duplicate acks.

Que algoritmo utiliza o emissor para adaptar o envio de segmentos ao congestionamento na rede?
- 3 Componentes do algoritmo
    - AIMD (additive-increase, multiplicative-decrease
    - Slow start

- Reação a eventos timeout.

No **início** da conexão o **TCP** coloca cwnd=1 MSS.

Slow Start:

- Crescimento exponencial.
- O valor de cwnd duplica a cada RTT.
- Isto é feito aumentando o cwnd de 1 MSS para cada ack que chega.
- Feito pois pode existir muita largura de banda livre, e seria ineficiente um aumento linear nesta fase inicial.
- Quando ocorre uma perda, isto para e:
  - Se a perda for detectada por 3 duplicate ACKs – cwnd passa para metade e depois cresce linearmente.
  - Se a perda for detectada por timeout – Entra em slow start (cwnd=1 MSS) e cresce exponencialmente até atingir metade do valor anterior. Depois cresce linearmente.

Notas 8

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 conjunto de nós da rede)

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**.

O papel da **camada de rede** é mover pacotes de um host emissor para um host receptor.

Funções da camada de rede:

- **Forwarding**
  - Quando chega um pacote é necessário mover o pacote do link/physical port de input para o link/physical port de output.
  - Feito com base em tabelas com determinadas informações.
  - Acção local.
  - Um pacote pode ser bloqueado (e.g., destino proibido) ou duplicado (enviado para vários links de saída).
- **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.

Podemos chamar "packet switches" aos **switches** e **routers**.

Os **switches** e **routers** tomam as suas decisões de **forwarding** com base num campo da camada link-layer.

Um router terá:

- Ports de input e output
- Uma forma de ligar os ports de entrada aos ports de saida
- Um processador de routing que executa os protocolos de routing, mantém tabelas de forwading e informações de routing.

Os input ports têm funções:

- Da camada física – Faz a transmissão e 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.
- Forwarding e Queueing - Muitos routers mantêm uma cópia da tabela de forwarding nos input ports por questões de rapidez.

Prefix Match

- Termina no digito a partir do qual todas as combinações de bits são aceites para essa entrada.
- 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.

O **output port** transmite os pacotes, que estão em buffer, pelo link de saida.

Protocolos de Routing – Protocolos que actualizão as tabelas e escolhem os caminhos.

Formato do Header IP:

- **Versão do Protocolo IP:**
  - Para que o router possa interpretar correctamente o que se segue.
  - O IPv6 tem um formato diferente do IPv4.
- **Header length:**
  - Para determinar onde começam os dados.
  - Podem existir opções no header. Se não existem opções o tamanho é de 20 bytes.
- **TOS (Type of Service):** Para classificar datagramas.
- **Datagram length**: Tamanho do pacote (header+dados) em bytes.
- **Identifier:**
  - Numera os datagramas
  - Vai sendo incrementado
  - Quando existe fragmentação, cada fragmento tem o identificador do datagram original, permitindo assim que o destino faça a reassemblagem.
- **Flags:**
  - As flags são três: 1 bit não é usado (reserved), bit DF (Don't Fragment), bit MF (More Fragments).
  - Se existir framentaçao, os primeiros fragmentos têm a MF=1, enquanto que o último fragmento tem MF=0.
  - Para forçar que não haja fragmentação coloca-se DF=1 (o ICMP pode, no entanto, reportar um erro caso o tamanho do datagrama seja superior ao MTU num determinado ponto da rede).
- **Fragment offset**: Permite que o destino junte os pedaços pela ordem certa.
- **TTL (Time To Live):**
  - Garante que os datagrams não circulam eternamente na rede devido a inconsistencias e loops nas tabelas de routing.
  - Decrementa 1 valor sempre que passa por um router
  - É "dropped" quando chega a 0.
- **Protocol:**
  - Apenas é usado quando o datagrama chega ao destino. Especifica o protocolo de transporte a quem deve ser entregue os dados (6=TCP,17=UDP). Análogo ao port number no header do segmento TCP/UDP. É o que liga as camadas.
- **Header checksum:**
  - Para detecção de erros. São feitas somas de blocos de 16 bits (2 bytes) e acha-se o complemento a 1 tal como descrito no checksum do TCP/UDP. Note-se que ao contrário do TCP/UDP em que o checksum é feito sobre todo o segmento, aqui o cheksum é só sobre o header.
- **Source and destination IP address:** endereços IP origem e destino.
- **Options**: Fazem variar o tamanho do pacote e o tempo de processamento nos routers.
- **Data**:
  - Segmento TCP/UDP ou outros dados como mensagens ICMP.
  - O campo "Protocol" identifica o que está nesta seção.

Nas interfaces **Ethernet** MTU=1500 bytes. Os datagramas com tamanho superior serão que ser **fragmentados**. O MSS (maximum segment size) deve ser definido de forma a serem evitadas fragmentações, o que aumento o desempenho.
Não é bom um MSS pequeno porque existe mais headers para a mesma quantidade de dados transportada.

Exemplo: Datagrama de 4000 bytes (20 bytes header IP + payload) e MTU de 1500. Payload de um datagrama pode ser no máximo 1480, dado que MTU=1500. Logo, no primeiro datagrama offset=0, no segundo datagrama offset=1480/8, no terceiro datagrama offset=(1480x2)/8. Como cada datagrama terá que ter um header IP, os tamanhos dos datagramas serão 1500 (20 bytes de header + 1480 de payload), 1500 (20 bytes de header + 1480 de payload) e 1040 (20 bytes de header + 1020 de payload). Isto é, a soma dos payloads (1480+1480+1020 dá 3980 que era o payload do datagrama original).

Cada interface em cada host/router tem de ter um IP único. Logo um router associado a uma interface pode possuir vários IPs.

Uma **subrede** inclui um conjunto de hosts que têm a parte de subrede no endereço IP igual.
Os **hosts** numa **subrede** estão interligados "**directamente**" (via hub ou switch) sem nenhum router pelo meio.

Exemplo Subredes:

- O endereço de uma subrede termina com 0s na parte de hosts.
- É usada a notação 223.1.3.0/24 onde /24 é chamada a máscara de subrede.
- A máscara indica quantos bits, a contar da esquerda, pertencem à subrede. Também poderia ser usada a máscara 255.255.255.0 (em binário temos 11111111.11111111.11111111.0)
- As posições onde estão 1s referem a parte de subrede

Numa **mascara /24**, o número de IPs que se pode atribuir a **hosts** é (2^8)-2:

- Retira-se o **IP da subrede** - A combinação de bits com 0s na parte de host (ex: 223.1.3.0/24)
- Retira-se o **broadcaster** - A combinação de bits com tudo 1s na parte de host (ex: 223.1.3.255/24).

Quando a parte de rede tinha de ter 8, 16 ou 24 bits, as redes pertencem às classes A, B e C respectivamente.

CIDR (Classless Interdomain Routing):

- Estrategia de routing.
- Formato: a.b.c.d/x

Tipicamente uma empresa recebe um bloco contiguo de endereços IP, tendo hosts sempre com um prefixo comum. Assim, os routers fora da empresa, apenas precisam de pequenas tabelas já que basta uma entrada a.b.c.d/x para encaminhar todos os datagramas destinados a essa empresa.
Dos restantes bits para atribuir a hosts, parte pode ainda servir para dividir a rede da empresa em partes: subnetting. Exemplo: a.b.c.d/21 poderá ser o endereço CIDR da empresa, e comum a todos os hosts, e internamente a empresa pode utilizar a.b.c.d/24 para se referir a uma subrede dentro da organização (3 bits para subredes).
Neste caso o ISP está a usar 3 bits para subnetting, que dá para fazer 2^3=8 subredes.

Broadcast propaga-se para todas as máquinas até ao router.

Como é obtido um IP?

- Configurado no software
- Via DHCP - Um cliente envia um broadcast a perguntar se existem servidores DHCP a "escutar". O servidor de DHCP apanha esse pedido e responde com uma oferta de IP.

Notas 9

NAT – Reescrever o endereço IP e o numero de porto usando tabelas de hash.
Firewall - Bloqueia o trafego com base nos campos dos headers.

SDN:

- Rede Definida por Software
- É uma abordagem de arquitetura de rede que permite programar a rede de maneira central e inteligente usando aplicativos de software.
- Ajuda as operadoras a gerenciar toda a rede de modo consistente e holístico, seja qual for a tecnologia de rede subjacente.

OpenFlow:

- Standard que permite escrever nas "flow table" dos switches e routers.
- Permite que equipamento de diferentes fabricantes, sejam geridos remotamente de forma igual (um único protocol aberto).
- A comunicação ocorre através de canais SSL/TLS.
- Protocolo para comunicação entre controlador e dispositivos.

O **forwarding tradicional** baseia-se no "match" do endereço IP de destino e numa "action" que é colocar o pacote numa porta de saída específica.

O **openflow** consiste num paradigma match+action mais genérico. O match pode ser feito com base em vários campos nos headers das várias camadas, e as actions podem ser várias.

Cada entrada na tabela de forwarding pode ter:

- Um conjunto de campos de headers, para o match.
- Ações, caso exista match (ex: enviar para 1 mais ports, fazer o drop, alterar campos do header, …). Caso contrario, pode ser dropped ou enviado para o controlador para processamento adicional (ex: avaliar possível adição de entrada).
- Contadores (ex: numero de pacotes em que ocorreu match, ultima vez que a entrada na tabela foi atualizada, …)

Switch Port - ID da porta física de entrada (por onde chegou o pacote).

Per-router control - Cada router faz as suas tarefas associada ao plano de controlo.

O **Sdn** faz as tarefas associadas ao plano de controlo dos vários switches no seu domínio.

Os hosts estão ligados a um router local e precisam que as tabelas tenham informação sobre qual o caminho (de preferencia o melhor) entre o router local na origem e o router local no destino. Assim sendo, são necessários algoritmos para encontrar estes caminhos.

Usando grafos para calcular o caminho:

- Um link tem normalmente um custo (peso) associado.
- O custo de um caminho é a soma dos custos dos links percorridos.
- Se custo é 1 em todos os links, o problema resume-se ao caminho com menos saltos (hops).
- Se custo está relacionado com a largura de banda de forma inversa: baixo custo significa que o link tem muita largura de banda disponível.

- Se custo está relacionado com o congestionamento de forma directa: baixo custo significa que o link tem pouco congestionamento.

Os **algoritmos de routing** podem possuir informação global de toda a rede, ou esta informação pode estar descentralizados. Os algoritmos podem também ser estáticos ou dinâmicos.

Algoritmos "Link-state" – Algoritmos que conhecem todos os links e seus custos.

Algoritmos "Distance vector" – Algoritmos que contêm um vector de custos estimados para todos os outros nós na rede.

Algoritmo de Dijkstra – Algoritmo para encontrar o caminho mais curto nos grafos.

Como evitar oscilações no Algoritmo de Dijkstra?

- Evitar que os routers corram o algoritmo Link State ao mesmo tempo.
- Randomizar o próximo instante em que o router irá enviar um "link advertisement" ao outros routers

Notas 10

Bellman Ford (5.21):

- O custo do caminho mais barato de u para z é a menor das somas do custo de um link vizinho com a distancia do vizinho a z.

Algoritmo Distance Vector:

- Dx(y). Cada nó começa com um valor estimado da distancia até a um nó y, para todo o y.
- O nó x sabe os custos para cada vizinho v: c(x,v)
- O nó x mantém os vectores distancia dos seus vizinhos
- Mensagens trocadas apenas entre vizinhos. O número de mensagens necessárias depende da convergencia do algoritmo.
- Se houver sabotagem, o problema é grave pois o host está a calcular tabelas que vai difundir.

Algoritmo Link State:

- Um nó necessita de conhecer os custos de todos os links.
- Sempre que existe uma alteração de um custo essa alteração tem de ser enviada a todos os nós, via broadcast.
- Se houver algum problema não é muito grave pois um host está a calcular apenas as suas tabelas, e não as vai difundir.

Normalmente, nem todos os nós de uma rede usam o mesmo algoritmo de routing, porque conforme o número de routers começa a crescer, torna-se impossível ou inviável armazenar nas tabelas de routing todos os routers destino e actualizar informações acerca de todos.

Autonomous Systems - Os routers estão sob o mesmo controlo administrativo. Os routers num **AS** correm todos o mesmo protocolo de routing.

Intra-AS Routing Protocol:

- Protocolo de routing que corre dentro de um AS.
- ASs podem correr protocolos intra-AS diferentes.

Gateway - Router de um AS que faz a ligação com outros ASs, isto é, para fora do seu AS.

O **ICMP** é considerado como parte do IP mas está acima do IP uma vez que as mensagens ICMP são transportadas dentro de datagramas, tal como segmento TCP ou UDP.

Formato de um pacote ICMP:

- type (8 bits)
- code (8 bits)
- checksum (16 bits)
- data (variavel, depende de type e code)

O programa **Traceroute** é implementado com mensagens **ICMP.**

Notas 11

Protocolos da Link Layer:

- Definem o formato dos pacotes trocados entre os nós nos extremos de um link.
- Os pacotes são chamados frames.

Serviços de um protocolo da camada de ligação:

- **Framing** – O datagrama é colocado dentro de um frame antes de os dados serem transmitidos. Isto significa colocar um header e nalguns casos um trailer (ambos são chamados "header fields").
- **Acesso ao meio:**
  - Poderá existir um protocolo MAC que impõe as regras de acesso ao meio partilhado.
  - O endereço MAC (Endereço físico) é usado no header da frame para identificar a fonte e o destino num meio partilhado.
- **Entrega fiável:**
  - Pode ser feita para links de elevada taxa de erros (ex:wireless).
  - Os links de fio muitas vezes não têm este serviço.
  - Funciona também com acks e retransmissões tal como visto na camada de transporte.
  - O erro é corrigido localmente, onde ocorre.
- **Possui flow control.**
- **Error detection -** O receptor pode detectar erros nos bits e solicitar ao emissor a retransmissão. A correção de erros (apenas no header) é também possível.

A **camada de ligação** é implementada numa **placa de rede,** também conhecida como **NIC** (Network Interface Adaptor).

O **NIC** é semi-autonomo no sentido que pode desprezar uma frame com erros sem que outros componentes saibam. Quando tem dados gera uma interrupção para avisar o host/router.

Técnicas da camada de ligação para correção e deteção de erros:

- Na origem são colocados alguns bits para deteção de erros (EDC) para além dos dados (D).
- No destino verifica-se se os recebidos são iguais aos enviados.

Métodos para deteção de erros:

- **Paridade:**
  - Se usada paridade par, o bit extra deverá ser tal que o numero total de bits a 1 é par.
  - Desvantagem: Fornecem pouca proteção se houver troca de mais que um bit
- **Checksumming:**
  - Os bits dos dados são tratados como sequencias de k bits. Um método é somar as várias sequencias de k bits e usar o complemento a 1 desta soma como informação de checksum.
  - Vantagem: Colocam pouco overhead no seu cálculo.
  - Desvantagem: Fornecem pouca proteção contra erros.
- **CRC (cyclic redundancy checks) 6.14;**

Tipos de links:

- **Point-to-point:** Liga um emissor a um receptor. Exemplo de um protocolo: PPP (são usadas linhas telefónicas).
- **Broadcast:** Múltiplos emissores e múltiplos receptores ligados a um canal partilhado. Exemplo de protocolo: Ethernet, wireless LANs.

Nos protocolos **MAC** (Medium Access Control) a troca eventual de informação, que é necessária para determinar como o canal é partilhado, é feita através do canal de dados (in-band)

Ethernet - Quando os hosts estão ligados a um hub que faz broadcast, este funciona como um protocolo do tipo CSMA/CD.

Slotted ALOHA:

- Tempo dividido em slots iguais
- Todos os frames possuem mesmo size
- Se assumirmos que p é uma probabilidade entre 0 e 1:
  - Quando um nó tem uma frame "fresquinha" para transmitir, transmite no próximo slot.
  - Se existe colisão o nó retransmite a frame, em cada slot seguinte, com probabilidade p até conseguir enviar a frame.

Desvantagens de Slotted ALOHA:

- Caso exista colisão o slot é desperdiçado.
- Como a retransmissão é feita com probabilidade p, podem existir slots "idle" embora existam frames para enviar.
- A deteção de colisão poderá ser feita muito antes de completar o slot de transmissão de um pacote e essa possibilidade não é usada para aproveitar melhor a largura de banda. Só no fim do slot, inicio do próximo slot, é que ocorre nova transmissão.

Nos protocolos **ALOHA** a decisão de enviar uma frame é independente da actividade dos outros nós. No **CSMA** a acção dos outros nós é tomada em consideração.

Quanto mais longa a distancia e atraso de propagação maior a probabilidade de **colisões**.

CSMA/CD - CSMA com deteção de colisões.

Porque as wireless LANs não usam CSMA/CD?
A capacidade de deteção de colisões exige que a estação consiga enviar e receber ao mesmo tempo. Como a força do sinal recebido é muito fraco (dado o meio de transmissão) quando comparado com o sinal enviado, torna-se muito caro conceber hardware que detecte colisões.

Jam signal - 48 bits que correspondem a frame mal formada para que todos detectem.

Eficiência (6.34) - Fração de tempo em que está a haver transmissão produtiva (sem colisões) quando o numero de estações é grande e tem muitas frames para enviar.

Quando o **tempo de transmissão** da frame aumenta a **eficiência** aumenta porque a transmissão está a ser muito produtiva e assim todas as outras estações não transmitem porque estão a escutar o meio.

TDM e FDM - Ineficiente para baixas taxas mas eficiente e justo se todos estão a transmitir.

CSMA - Eficiente para baixas taxas, porém possui muitas colisões (ineficiente) para taxas altas.

Taking turns – Junta-se o melhor do CSMA e de TDM e FDM.

Notas 12

Protocolo ARP – Descobre o endereço MAC associado ao IP de um gateway.

Ethernet é "unreliable" e "connectionless".

Quando o meio é partilhado o **Ethernet** usa **CSMA/CD**.

Não é possível com um hub ligar Ethernets de velocidades diferentes porque o Hub não faz buffering da frame para posterior transmissão da frame com outra codificação e maior numero de transições de bits.

Um switch já permite que Ethernets diferentes comuniquem entre si porque é um dispositivo da camada 2, armazena a frame e examina o header da frame. Tem uma tabela MAC e, por isso, pode verificar o endereço destino na frame e enviar a frame para o porto correcto sem difundir para os restantes portos.

Hub: Um só domínio de colisão.
Switch: Cada segmento um domínio de colisão. Isto é, frames para segmentos diferentes não colidem.
Router: Quando necessitamos de subredes distintas ou acesso ao exterior (outras subredes) temos de ter um router.

Os routers isolam tráfego, controlam broadcasts e encaminham.

Hubs, switches e routers são todos dispositivos de interconexão.