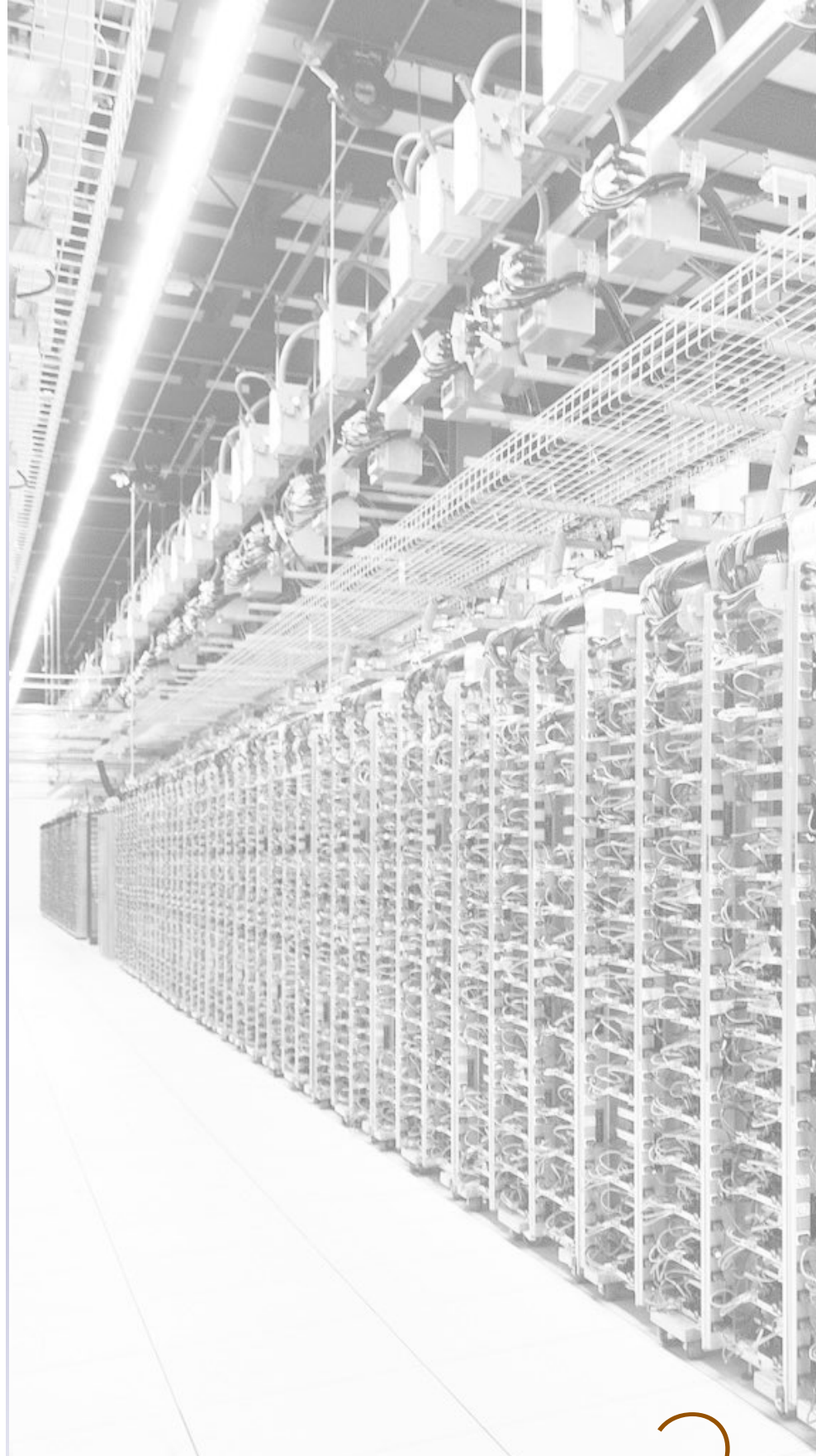




universidade de aveiro
theoria poiesis praxis

One of the things I like about the computer that I use is that I can write a program on it or I can download a program on to it and run it. That's kind of important to me, and that's also kind of important to the whole future of the internet... obviously a closed platform is a serious brake on innovation.

Tim Berners-Lee



3

FUNDAMENTOS DE REDES

Atenção!

Todo o conteúdo deste documento pode conter alguns erros de sintaxe, científicos, entre outros... **Não estude apenas a partir desta fonte.** Este documento apenas serve de apoio à leitura de outros livros, tendo nele contido todo o programa da disciplina de Fundamentos de Redes, tal como foi lecionada, no ano letivo de 2015/2016, na Universidade de Aveiro. Este documento foi realizado por Rui Lopes.

mais informações em ruieduardofalopes.wix.com/apontamentos

Todos os dias, milhões, senão milhares de milhões de pessoas ligam os seus computadores pessoais ou *smartphones* e acedem a uma rede global que nos permite efetuar comunicações eficientes e eficazes - a Internet. Nesta disciplina iremos estudar o efeito e a produção de redes com escalas muito inferiores à Internet, contudo, tomando as bases como o essencial deste nosso estudo, iremos aprofundar conhecimentos em termos de fundamentos de redes.

1. Redes de Computadores e a Internet

Hoje a Internet é, sem sombras de dúvida, o maior sistema algumas vez criado pela humanidade com centenas de milhões de computadores ligados entre si, com milhares de milhões de utilizadores em funções 24 horas por dia, 7 dias por semana. Nesta disciplina, como já foi referido, ir-nos-emos focar em particular nesta grande rede que existe e em particular em parte dos protocolos que nela existem e nas redes que a incorporam. Mas afinal o que é a Internet? A **Internet** é então um rede produzida por milhares de milhões de dispositivos interligados a que nós iremos denominar de **hosts** ou de **sistemas terminais**. Os sistemas terminais, por sua vez, estão ligados por vias de comunicação e **switches** - os switches são equipamentos de redes que permitem distribuir uma ou mais redes segundo uma política estabelecida numa tabela de redirecionamento, como teremos oportunidade de verificar mais à frente.

Esta grande rede possui diferentes linhas de comunicação, o que provoca que todas tenham taxas de transferência diferentes, medidas em bits por segundo.

Internet
hosts
sistemas terminais
switches

Transmissão de informação e introdução ao protocolo IP

Quando um sistema terminal precisa de enviar informação para outro terminal é feita uma segmentação da informação, dividindo o produto a ser enviado em várias partes, enviando **pacotes**, adicionando, também, cabeçalhos de introdução (em inglês denominados de **headers**). No final, quando os pacotes são transmitidos, através da informação contida em cada um dos pacotes (nos cabeçalhos), o sistema terminal deve ser capaz de montar tudo de novo, reavendo o produto inicial.

pacotes
headers

A transmissão dos pacotes é assegurada, principalmente, por um equipamento especial, chamado de **router**. Um router é um equipamento que visa distribuir pacotes segundo políticas mais rígidas. Consideremos assim, a título de exemplo, que temos uma rede composta de um sistema terminal ligado a um router, este, ligado a outro sistema terminal, conforme representado na Figura 1.1.

router

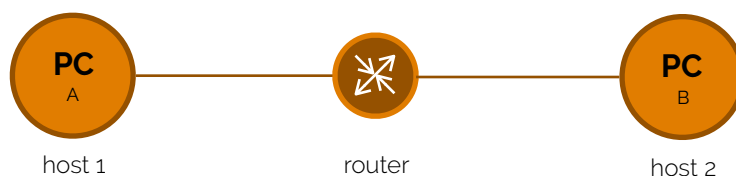


figura 1.1
hosts e router

Na Figura 1.1 imaginemos que pretendemos transmitir um ficheiro do computador pessoal A para o computador B. Para que isto aconteça um ficheiro sai do host 1 todo fragmentado em pacotes. Mal chega ao router o pacote é distribuído para o próximo caminho até ao host 2. Quando chega ao host 2, os pacotes são novamente montados de acordo com os cabeçalhos disponíveis e o conteúdo do ficheiro passa a estar disponível no computador pessoal B. Mas o que é que acontece se a ligação entre o router e o host 2, da Figura 1.1, se quebrar? Caso isto aconteça, o ficheiro sai em pacotes do host 1, chegam ao router e, se não houver outro caminho senão o mesmo por donde veio, todos os pacotes serão desperdiçados.

Mas o que é que permite que esta transferência na rede aconteça sempre nos mesmos conformes que os explícitos no raciocínio acima? Introduzimos assim o **protocolo**

IP. Este protocolo, sigla de *Internet Protocol* é o responsável por assegurar que os processos de transferência numa rede, nesta camada, se conduzam sempre da mesma forma.

protocolo IP

Endereçamento IP (versão 4)

As transferências numa rede são análogas ao sistema de correios de uma região ou país. Consideremos assim que nós pretendemos enviar uma carta para um dado destinatário - o que é que precisamos para que isto se concretize? Depois de escrita uma mensagem, precisamos de saber para quem mandamos ao certo e como é que mandamos. Para mandar uma carta é simples - basta saber a morada para quem queremos enviar a carta. Mas como é que isto pode ser feito em termos de Internet? Da mesma forma que com cartas precisamos de uma morada, em termos de redes precisamos de um **endereço**. O protocolo IP, numa versão 4, estabelece várias formas de produzir endereços, aos quais chamar-lhe-emos **endereços IPv4** (o "v4" provém do facto de estarmos a estudar a versão 4 do protocolo IP).

endereço

endereços IPv4

Os endereços IPv4 dividem-se por classe, cada uma definindo uma relação aplicável entre o número de redes e o número de terminais às quais podem corresponder. Esta designação é, no fundo, um identificador único de uma interface de rede, tendo 32 bits de comprimento, identificando uma rede (com um código de *netid*, também chamado de **prefixo de rede**) e um terminal (com um código de *hostid*). Na Figura 1.2 podemos ver as diversas classes sobre as quais os endereços IP se regem.

prefixo de rede

figura 1.2
classes de IP

0	7			15			23			31				
0		netid			hostid								classe A	
1		0		netid				hostid					classe B	
1		1		0		netid					hostid		classe C	
1		1		1		0		endereço multicast						classe D
1		1		1		1		reservado para utilização futura						classe E

O endereçamento, em termos de protocolo, pode ser de um de quatro tipos: **unicast**, se se identifica um único recetor (host); **broadcast**, se todos os hosts presentes são recetores; **multicast**, se se identificam todos os elementos de um determinado grupo como recetores (*all-of-many*); ou **anycast**, se se identifica um elemento de um grupo como recetor (*one-of-many*). Na Figura 1.3 podemos ver um exemplo para cada um destes endereçamentos, sendo (a) unicast, (b) broadcast, (c) multicast e (d) anycast, representando *E*, como emissor.

unicast, broadcast
multicast
anycast

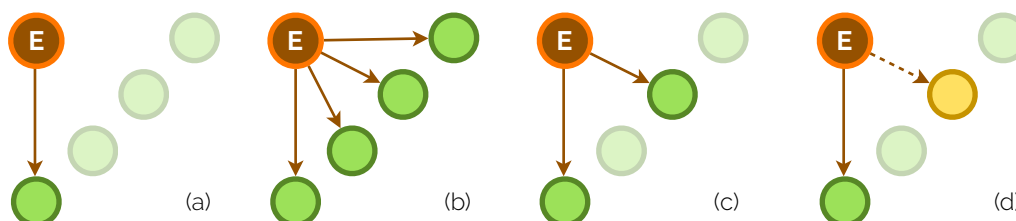


figura 1.3
unicast, broadcast, multicast
e anycast

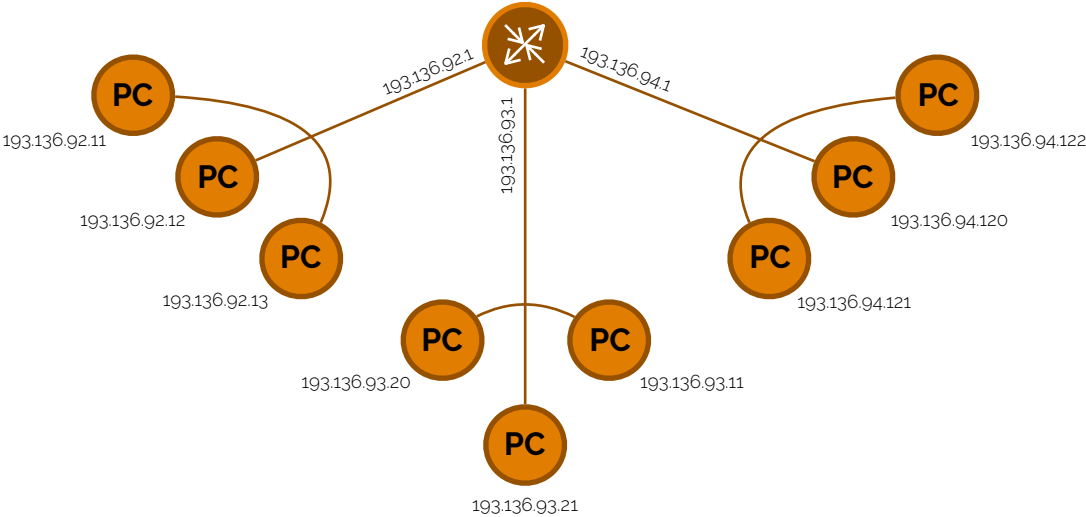
Em termos de notação, usa-se uma **notação** decimal com pontos, cujos bytes incluídos são identificados por um número decimal de 0 a 255. Por exemplo, um IP válido será o endereço 192.136.92.1 = 11000001.10001000.01011100.00000001 (em binário).

notação

Uma **interface** é uma ligação entre um sistema terminal/router e a rede. Consideremos a Figura 1.4. Como podemos verificar, os routers, aqui representados por um círculo com setas, têm, tipicamente, mais de uma interface (neste caso os endereços 193.136.92.1, 193.136.93.1 e 193.136.94.1). Já os hosts, ou terminais, têm, normalmente, uma interface (mas podem ter mais de uma). Com isto, podemos concluir que cada interface tem de ter, pelo menos, um endereço IP associado, também podendo ter mais do que um.

interface

figura 1.4



Alguns endereços IP são especiais, pelo que não podem ser usados em qualquer situação, fora ocasiões especiais. Entre esses endereços, vejamos quais são e especifiquemos a sua função com algum detalhe: o endereço **0.0.0.0** significa "este terminal" e é apenas válido no decorrer de algumas inicializações; o endereço composto por 0's + hostid significa "host na própria rede" e é, também, apenas válido para algumas inicializações; o endereço 127.(...), sendo o mais comum 127.255.255.255 significa **loopback** e serve para simular comportamentos de *sockets*, pois tudo o que passa por este endereço não chega a tocar na rede; o endereço composto por netid + 1's significa "broadcast dirigido à rede identificada pelo netid" e não pode ser usado como endereço de origem; o endereço **255.255.255.255** significa "broadcast local" e transmite tudo o que recebe para todos os dispositivos ligados à rede, não podendo ser usado como endereço de origem; finalmente, o endereço composto por netid + 0's é um "identificador da rede netid", o que se usa apenas com designação.

0.0.0.0

loopback

255.255.255.255

Tal como vimos na Figura 1.2, os endereços IP estão divididos por classes, de forma a classificar o **espaço de endereçamento**: a classe A tem um prefixo de 8 bits, começando por 0₂; a classe B tem um prefixo de 16 bits, começando por 10₂; e a classe C tem um prefixo de 24 bits, começando por 110₂. A partir do ano de 1993 o tamanho do prefixo passou a ser definido pela máscara de rede. Na Figura 1.5 podemos ver a divisão do espaço de endereçamento.

espaço de endereçamento

classe	menor endereço	maior endereço	bits de prefixo	nº máximo de redes	bits de sufixo	nº máximo de hosts por rede
A	10.0.0	126.0.0.0	7	128	24	16777216
B	128.0.0.0	191.255.0.0	14	16384	16	65536
C	192.0.0.0	223.255.255.0	21	2097152	8	256
D	224.0.0.0	239.255.255.255				
E	240.0.0.0	255.255.255.254				

figura 1.5
divisão do espaço de endereçamento

Máscaras de Rede (IP versão 4)

Um componente importante na instauração de redes é o desenvolvimento de **máscaras** para o protocolo IP. Com a versão 4 do IP (IPv4) a máscara é utilizada para separar a parte da rede da parte da interface dos endereços, pelo que a parte da máscara (em binário), cujo bit seja 1 indica que esse bit no endereço faz parte do identificador da rede. Por outro lado, os bits da máscara que sejam 0, indicam que esses bit no endereço fazem parte do identificador do terminal (host).

Inicialmente os endereços IP tinham uma fronteira fixa entre o identificador da rede e da interface, definida a partir dos primeiros bits dos endereços e da máscara de tamanho pré-definido. Estas máscaras permitiram assim a criação de **sub-redes** (em inglês, *subnets*). Na Figura 1.6 podemos ver dois exemplos da aplicação de máscaras em dois endereços IP de classes diferentes.

máscaras

sub-redes

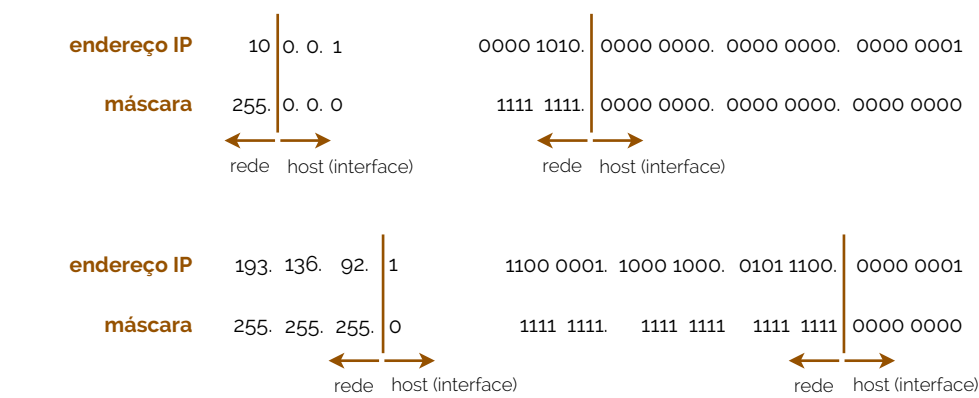


figura 1.6
máscaras de rede

Datagramas IP

O protocolo IP envia informação sob a forma de **datagramas**. Por cada bloco de dados (em bytes) que é enviado pelo presente protocolo, este adiciona um cabeçalho formando assim um datagrama IP. Cada datagrama (composto por um campo de cabeçalho e um campo de dados) é entregue ao MAC transportando uma camada específica para a rede. Em cada camada da rede física, uma outra com um frame produzido do MAC é composto de um cabeçalho do protocolo MAC e de um campo de dados, num processo denominado de **encapsulamento**.

O datagrama IP tem um determinado formato. Na Figura 1.7 podemos ver uma representação algo fiel de um datagrama IP.

datagramas

encapsulamento

0	4	8	16	19	24	31
VERS		HLEN	SERVICE TYPE		TOTAL LENGTH	
IDENTIFICATION				FLAG	FRAGMENT OFFSET	
TIME TO LIVE		PROTOCOL		HEADER CHECKSUM		
SOURCE IP ADDRESS						
DESTINATION IP ADDRESS						
IP OPTIONS (IF ANY)					PADDING	
DATA						

figura 1.7
datagrama IP

No datagrama IP, como podemos ver na Figura 1.7, existem muitos campos que merecem a nossa atenção. Assim, é importante saber que um cabeçalho de um datagrama

IP tem, no mínimo, 20 bytes, sendo que esse tamanho pode variar consoante haja opções designadas no cabeçalho, podendo crescer para 24, 28, 32, 36, ... Assim, temos:

- **VERS** - significando "versão", em 4 bits, especifica-se a versão do protocolo IP (versão 4, para o nosso contexto);
- **HLEN** - significando "*header length*", em 4 bits, especifica-se o tamanho do cabeçalho em blocos de 4 octetos, pelo que quando não tem nada do campo opcional (**IP OPTIONS**), o cabeçalho ocupa 5 blocos de 4 octetos e o primeiro assume o valor 0x45;
- **SERVICE TYPE** - significando "tipo de serviço", em 1 byte, especifica-se qual o tipo de serviço ao qual o pacote pertence (identifica para diferenciar o tratamento dos pacotes pelos routers com base na qualidade do serviço pretendida - por defeito este campo tem o valor 0x00);
- **TOTAL LENGTH** - significando "tamanho total", em 2 bytes, este campo deve destacar o tamanho total do datagrama IP em octetos, incluindo o cabeçalho, sendo que o tamanho máximo é de 65535 octetos e, no entanto, este tamanho está restringido pelo Maximum Transport Unit (**MTU**) da rede - mecanismo o qual é responsável pela fragmentação e pelo reagrupamento de pacotes;
- **IDENTIFICATION** - significando "identificação", em 2 bytes, especifica um identificador atribuído pela estação que gerou o datagrama, sendo que este campo é mantido durante o processo de fragmentação, permitindo o destinatário identificar vários fragmentos de um mesmo pacote;
- **FLAGS** - em 3 bits, o primeiro está reservado para uso futuro, assumindo sempre o valor 0, o segundo bit assume o valor 0 se o datagrama puder ser fragmentado e o valor 1 caso contrário, e o terceiro e último bit assume o valor 0 se for o último fragmento e 1 se não for;
- **FRAGMENT OFFSET** - significando "deslocamento do fragmento", em 13 bits, define qual o deslocamento na montagem dos pacotes, pelo a posição encontra-se em múltiplos de 8 bytes;
- **TIME TO LIVE** - significando "tempo de vida", em 1 byte, marca o tempo máximo que o datagrama pode permanecer em rede, sendo alterado em cada router e quando atinge 0 é eliminado, dado que este decrementa uma unidade ou no número de segundos que demorou a processar o datagrama;
- **PROTOCOL** - significando "protocolo", em 1 byte, permite especificar sob que tipo de protocolo o datagrama se rege;
- **HEADER CHECKSUM** - campo de verificação de 2 bytes dos outros campos, pois dado que o cabeçalho é alterado em cada router, este valor também é recalculado, e este campo permite assim, detetar erros de transmissão que alterem o cabeçalho do datagrama.

MTU

Address Resolution Protocol (ARP)

Consideremos a situação ilustrada na Figura 1.8, onde um router está ligado a três terminais, neste caso, computadores pessoais (A, B e C).

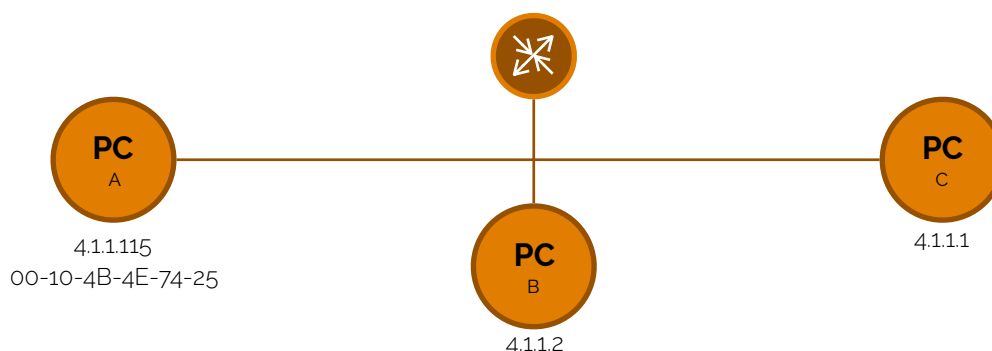


figura 1.8

Consideremos agora que o computador pessoal A (PC-A) pretende enviar um pacote para o computador pessoal C (PC-C). Dentro da rede, de momento, o computador A só conhece os endereços IP de cada um dos computadores e do router e o seu próprio endereço MAC. Mas o que é o **endereço MAC**? O endereço MAC, protocolo com o acrónimo de *Media Access Control* é um identificador único designado para uma dada interface numa rede num segmento físico. Mas para que o computador A possa enviar um pacote para o computador C, ele precisa do MAC de origem (que já o tem, dado que é o do próprio) e o MAC do destino (endereço físico do PC-C, que não tem), mais o IP de origem e de destino.

endereço MAC

Mas como é que o computador, sabendo o IP de destino, fica a saber o endereço MAC do computador C? Para tal existe um protocolo denominado de **ARP**, acrónimo de *Address Resolution Protocol*. Para saber o MAC o PC-A envia um **ARP Request** (pedido ARP) o que tem o seguinte efeito na rede, como mostra a Figura 1.9: um pedido surge do computador A para todos os dispositivos inseridos na rede até aos routers (não passando para além de nenhum router) perguntando para quem tem, neste caso, o IP 4.1.1.1, que se acuse.

ARP

ARP Request

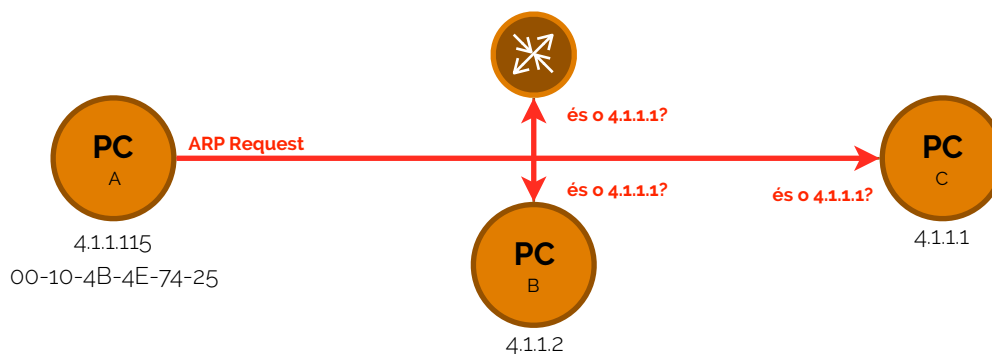


figura 1.9

ARP Request

Para completar os procedimentos deste protocolo, o dispositivo que cumpre os requisitos do ARP Request responde afirmativamente fazendo um **ARP Response** acompanhado do seu endereço MAC, como podemos ver na Figura 1.10.

ARP Response

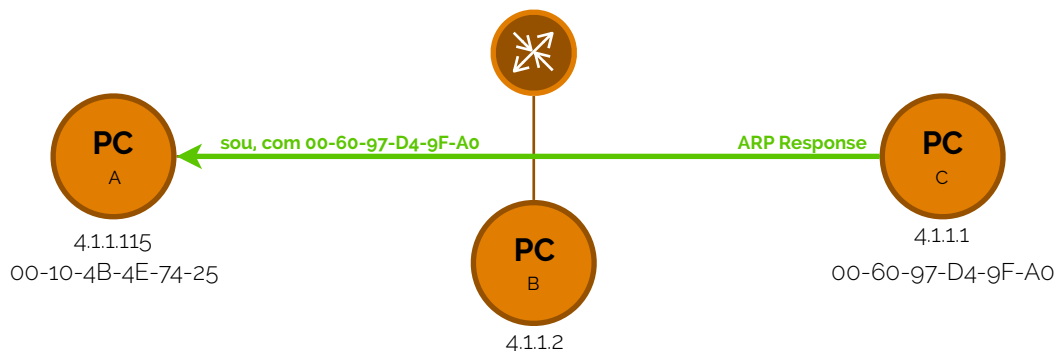


figura 1.10

ARP Response

Feito o ARP Response, agora o computador A tem todos os requisitos necessários para enviar um pacote para o PC-C.

Os endereços MAC são então endereços físicos das máquinas associadas às redes, sendo uma sequência de 48 bits, que normalmente fazem-se representar por pares de dígitos hexadecimais separados por hífenes (-) ou por dois pontos (:).

Default gateway

A ligação entre componentes de redes muitas vezes pode significar mudanças de contexto, como já tivemos oportunidade de referir anteriormente. Consideremos o exemplo de aplicação da Figura 1.11, onde podemos ver um terminal ligado a um router, este ligado sequencialmente a mais dois routers, por fim, ligando a um segundo computador.

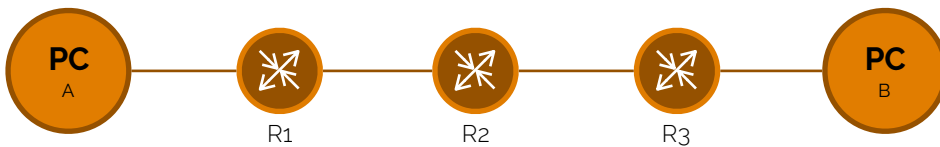


figura 1.11

Uma questão pertinente a fazer a respeito da Figura 1.11 é se ambos os computadores (PC-A e PC-B) são da mesma rede. Não considerando conceitos de redes privadas (a estudar mais à frente), que endereços é que podemos dar ao PC-A e ao PC-B? Para respondermos a estas duas questões temos de ter presente o conceito de router - o router é um equipamento que faz uma conexão entre redes. Assim sendo, podemos já responder à primeira questão: o PC-A e o PC-B não são da mesma rede, porque entre eles existem 3 routers, e entre cada dois existe uma rede. Quanto à segunda questão, nós não podemos, não sendo da mesma rede, dar endereços da mesma rede a ambos os computadores. Assim sendo é impossível, dadas as nossas considerações, dar o endereço 192.1.1.3/24 ao PC-A e o endereço 192.1.1.2/24 ao computador B. Uma forma de resolver este conflito é atribuindo o endereço, por exemplo, 10.1.1.1/8 ao computador B. Consideremos assim, atualizando a Figura 1.11, a Figura 1.12.

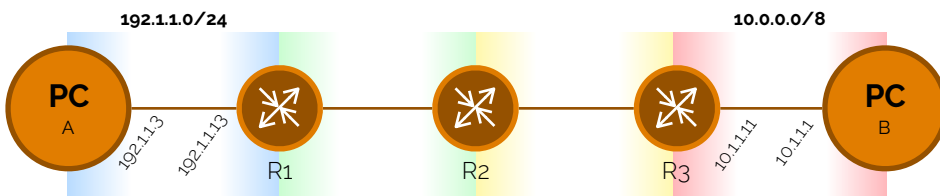


figura 1.12

Estando no computador pessoal A eu posso pretender enviar um pacote para o computador B, mas como é que isto pode acontecer? Para fazer isto precisamos que se faça, por exemplo, um ping no computador A para o computador B (ping 10.1.1.1). Se experimentarmos este cenário, sem quaisquer outras configurações realizadas, o que acontece é que, no terminal, a seguinte mensagem, de (1.1) aparecerá.

```
$ ping 10.1.1.1
```

(1.1)

```
Destination Host Unreachable
```

Então mas porque é que o destino não é acessível? Acontece que, tal como afirmámos anteriormente, o endereço 10.1.1.1 não pertence à mesma rede que o computador pessoal A, assim sendo, este não conhece outra rede que não a sua, logo não sabe como levar o pacote até ao computador pessoal B. Para resolver este tipo de situação existe uma configuração que define, caso um determinado terminal não conheça um dado destino, que todos os pacotes (nesta situação) passam por um caminho em específico. Define-se assim o **default gateway**. Neste caso em específico, um default gateway possível é o endereço 192.1.1.13. Dado isto, agora o pacote, pelo menos, será capaz de navegar até ao router 1 (R1), da forma que estudámos antes. Assim, o PC-A, se não souber o endereço MAC de R1 faz um ARP Request obtendo, num ARP Response, o MAC do R1. Só agora é que todos os campos necessários na trama Ethernet estão devidamente preenchidos.

default gateway

Mas será que com estas especificações o pacote viaja até ao computador B? Vejamos que a trama Ethernet contém o endereço IP de origem (IP da máquina PC-A), endereço IP de destino (IP da máquina PC-B), endereço MAC de origem (MAC da máquina PC-A), mas o endereço de destino MAC não é o endereço MAC do computador B. O que aqui acontece é que, mal o pacote chega ao router 1 (R1), se R2 não estiver já mapeado no encaminhamento do R1, este terá de fazer um ARP, de forma a conseguir definir caminho até R2. Da mesma forma, R2 fará ARP para R3, caso R2 não conheça já R3 (igualmente entre R3 e PC-B). Quando o pacote chega ao PC-B, a trama Ethernet contém o MAC de origem da máquina R3, MAC de destino da máquina PC-B, IP de origem da máquina PC-A e IP de destino da máquina PC-B. Na Figura 1.13 podemos um

resumo de toda a comunicação entre os equipamentos de um grafo no transporte de um pacote ICMP.

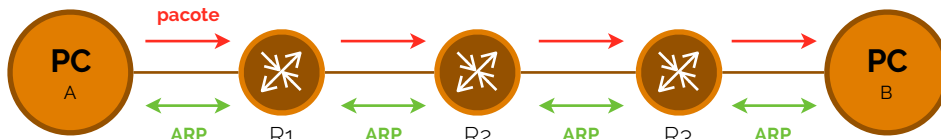


figura 1.13

Mas consideremos o caso da Figura 1.14. Aqui, um ficheiro que chegue pela rede 47.3 pode seguir vários caminhos até à rede 47.1. Como é que os routers escolhem o caminho mais acertado?

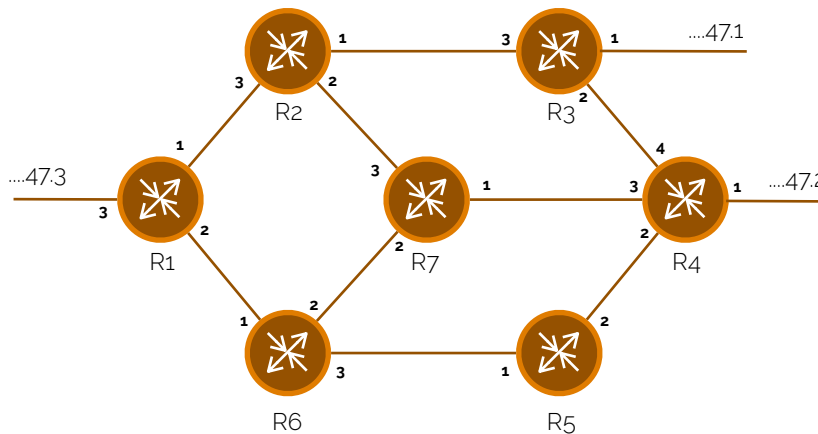


figura 1.14

Um pacote A que chegue à rede 47.3 e pretenda seguir caminho até à rede 47.1, mal se encontra prestes a sair da rede 47.3, encontra, no router, duas rotas possíveis (excluindo a de regresso). Aqui o router é o responsável por atuar sobre o pacote e capacitá-lo de privilégios para passar por um dos caminhos disponíveis. Para tal, cada router possui uma **tabela de encaminhamento IP**. Por exemplo, a tabela de encaminhamento do router 1 pode ser: para 47.1 → 1; para 47.2 → 2; para 47.3 → 3. Como podemos ver, para além de outros parâmetros que por enquanto ocultamos, as tabelas de encaminhamento têm pelo menos duas colunas - uma com o endereço de destino e outra com o caminho para si designado. Assim, como o pacote pretende ir para a rede 47.1, segundo a tabela, deve seguir pelo caminho legendado com o número 1. Se a tabela de encaminhamento for igual, neste caso, para os routers R2 e R3, então o pacote faria o percurso desenhado na Figura 1.15.

tabela de encaminhamento IP

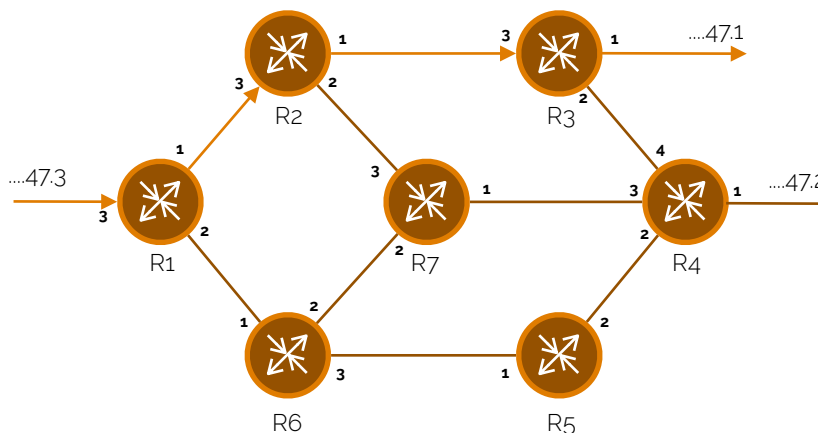


figura 1.15

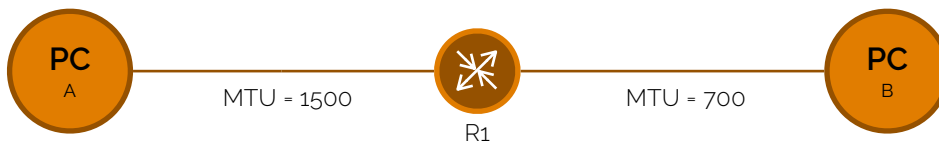
Fragmentação e montagem IP

Numa rede, existem várias variáveis que podem condicionar o tráfego das comunicações, entre os quais as capacidades físicas das linhas condutoras entre sistemas. Uma forma digital de condicionar uma capacidade de uma destas linhas é definindo um limite máximo de bits de uma transferência de dados. A este valor damos o nome de **MTU**, sigla de *Maximum Transfer Unit* (em português, Unidade Máxima de Transferência). Mas que consequências podemos obter das variações dos valores de MTU? Tendo o protocolo Ethernet um MTU de 1500, se tivermos um valor mais baixo que este, então teremos uma velocidade de ligação mais baixa, isto porque o pacote enviado, com um tamanho superior ao valor do MTU, terá de ser enviado por partes. A este último processo designaremos de **fragmentação**. Consideremos assim a Figura 1.16, onde se encontra um computador pessoal A ligado a um router (ligação com MTU de 1500) e, consecutivamente, a um computador pessoal B (ligação com MTU de 700).

MTU

fragmentação

figura 1.16



Tendo em conta a Figura 1.16 experimentemos enviar, então, um pacote ICMP de 900 bytes de PC-A para PC-B. Quando o pacote sai do computador A, como $900 < 1500$, este passa intacto até o router R1. A mesma situação não acontece para a frente do router R1. Como o MTU é de 700 bytes, entre R1 e PC-B, e nós temos 900 bytes para transmitir, nós teremos de fragmentar o nosso pacote. Fragmentamos então em duas partes, dado que 900 bytes não chega ao dobro do MTU. Na Figura 1.17 podemos então ver como é que os fragmentos se apresentarão.

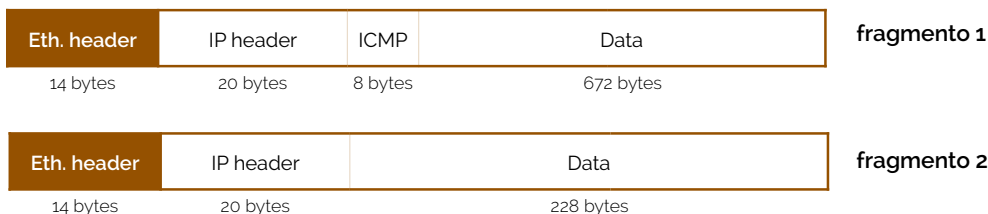


figura 1.17

fragmentação IP

Como podemos ver, pela Figura 1.17, o pacote ICMP que enviámos foi dividido em duas partes, mas tentemos perceber como é que o processo de fragmentação ocorreu. Se formos verificar o tamanho da trama do fragmento 1 rapidamente podemos ver que é de $700 + 14$ bytes. Então mas se o máximo estabelecido pelo MTU é de 700 bytes, como é que podemos enviar 714 bytes? Acontece que os 14 bytes excedentes fazem parte de uma secção da trama Ethernet, denominada por cabeçalho (daí a notação *Eth. header*, de *Ethernet header*). Os 700 bytes a que o MTU se refere correspondem aos dados contidos na trama Ethernet, isto é, a todos os campos que preenchem o corpo da trama Ethernet (IP header, ICMP e Data, no caso do fragmento 1).

No fragmento 1 temos, primeiramente, um cabeçalho IP¹ que nos identifica o tipo de protocolo IP a que o pacote se rege, qual o deslocamento do fragmento (através do campo *FRAGMENT OFFSET*), qual é o número de identificação (campo *IDENTIFICATION*) e quais são as flags, indicando, por exemplo, se este é o último fragmento de um pacote ou não (flag *MORE FRAGMENTS*). Seguidamente, temos um campo de 8 bytes denominado *ICMP* que serve para identificar o tipo de pacote em questão e um campo de dados, que contém apenas 672 bytes dos 900 que queremos que cheguem ao PC-B. Então mas porque é que o primeiro fragmento não tem 14 bytes de cabeçalho Ethernet + 700 bytes de dados? Acontece que é necessário, para além de identificar o fragmento, identificar o pacote,

¹ Note-se o cabeçalho IP pode ter mais de 20 bytes se o seu campo de opções estiver preenchido.

verificar se é o último fragmento, verificar se não houve erros na transmissão, ... e isso é garantido pela transmissão das tramas completas de IP e do pacote, neste caso, ICMP.

Isto tudo só nos deixa com uma última questão: porque é que o fragmento 2 não tem, então, o campo ICMP? Esta é uma questão muito normal, mas o que acontece é que os fragmentos, como já dissemos anteriormente, estão todos identificados pelo mesmo campo IDENTIFICATION, pelo que se já se fez corresponder o tipo do pacote num dos fragmentos então será redundante copiar o mesmo campo em todos os campos - nós precisamos da melhor gestão de espaço possível (enviar fragmentos extra pode ser dispendioso). Na Figura 1.18 encontra-se então, uma possível contextualização do exemplo da Figura 1.17, em termos de valores dos campos IDENTIFICATION, FRAGMENT OFFSET e MORE FRAGMENTS, do cabeçalho IP de cada um dos fragmentos.

fragmento 1		fragmento 2	
identification	385	identification	385
fragment offset	0	fragment offset	680
more fragments	1	more fragments	0

figura 1.18
fragmentos IP

Sub-redes

Uma **sub-rede** (em inglês, *subnet*) é um subconjunto de uma rede de classe A, B ou C. Consideremos assim o endereço IP 10.0.0.0/8. Como é que podemos criar subconjuntos desta rede? Através da utilização de máscaras de rede podemos estender a parte da rede à parte do terminal (host) do endereço IP, aumentando o número de redes e reduzindo, claro está, o número de hosts. Por exemplo, o endereço 10.32.0.1/11 usa os primeiros onze bits para definir o campo *netid* e os restantes 21 bits para definir o campo *hostid*. Esta é uma sub-rede da do endereço especificado anteriormente (10.0.0.0/8), de classe A. Vejamos com mais detalhe na Figura 1.19.

sub-rede


endereço IP	10. 32. 0. 1	0000 1010. 001 00000. 0000 0000. 0000 0001
máscara	255. 254. 0. 0	1111 1111. 111 00000. 0000 0000. 0000 0000
		

figura 1.19
sub-rede

Como exemplificado pela Figura 1.19, é possível também criar mais endereços desta mesma sub-rede, sendo um endereço desta mesma sub-rede se e só se tiver o campo do host todo a zeros + 1 até todo a uns + 0. Assim sendo, temos que a próxima sub-rede é a 10.64.0.0/11.

Consideremos um segundo exemplo e tentemos obter uma sub-rede da rede 192.228.17.0. A sub-rede deve ser de máscara 255.255.255.224, o que significa que teremos 1111 1111. 1111 1111. 1111 1111. 11100000₂, o que corresponde a 111 de sub-rede e 000000 de host, pelo que teremos 5 bits para "32 terminais". De facto não são 32 terminais, porque das 32 combinações possíveis, duas delas são restritas para casos especiais (192.228.17.0 e 192.228.17.31 - esta sub-rede é broadcast, respetivamente). Assim sendo, podemos dizer que a rede seguinte será a 192.228.17.32 (que tem endereços de 192.228.17.33 até 192.228.17.62), e assim sucessivamente.

2. Camada de Ligação (Data Link Layer) - I

Até ao momento estudámos a estrutura de ligação entre diferentes redes de forma muito conceptual - focámos-nos, principalmente, no protocolo IP. Consideremos agora que temos uma ligação como a representada na Figura 2.1. Nesta figura, do Router 1 para a esquerda temos uma ligação conjunta entre terminais, mas que no entanto, pertence à mesma rede (não há mais routers ligados no seu interior). A este aglomerado de terminais

e outros equipamentos numa só rede damos o nome de **rede de área local**, muitas vezes denominado simplesmente de **LAN** (do inglês *Local Area Network*). Até ao momento só trabalhamos com um tipo de tecnologia LAN - a Ethernet, com o protocolo **IEEE 802.3** -, mas há muitos outros protocolos que possuem a tecnologia LAN, como o IEEE 802.11 (Wireless) ou o IEEE 802.5 (Token Ring), entre outros.

rede de área local
LAN
[IEEE 802.3](#)

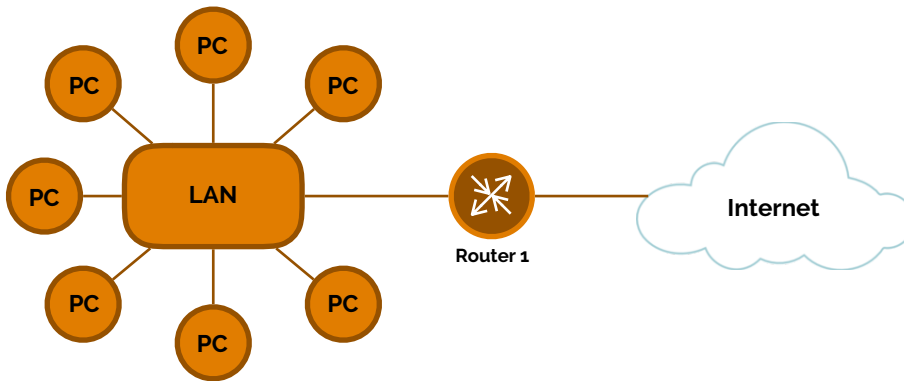


figura 2.1
LAN

Endereçamento em LANs (endereçamento IEEE)

As LANs estão implementadas a nível físico através de um adaptador denominado de **NIC** (acrónimo inglês *Network Interface Card*) que está ligado aos terminais, router e contém todos os parâmetros para poder estabelecer comunicações com todos os equipamentos ligados à mesma rede. Cada adaptador NIC possui um endereço único, o qual já referimos anteriormente: um **endereço MAC** (também muitas vezes designado de endereço físico ou de hardware, entre outros nomes...). Este endereço é gerido por uma entidade internacional de seu nome **IEEE** (acrónimo de *Institute of Electrical and Electronics Engineers*) e todas as suas tecnologias têm o mesmo esquema de endereçamento - comprimento de 48 bits e globalmente administradas pela IEEE. Estes endereços são depois codificados pelos fabricantes das NIC e assim se garante a unicidade de endereços.

NIC
endereço MAC
IEEE

Existem três tipos de endereçamento: unicast (identifica uma NIC), multicast (usado para comunicações multicast) e broadcast (um endereço especial usado para enviar frames para todos os terminais da sua rede).

Nos endereços IEEE, como é o caso do endereço MAC, os primeiros 3 bytes são reservados para designação pela IEEE e são chamados de **OUI** (*Organizationally Unique Identifier*). Na Figura 2.2 podemos encontrar um exemplo de endereço IEEE.

OUI

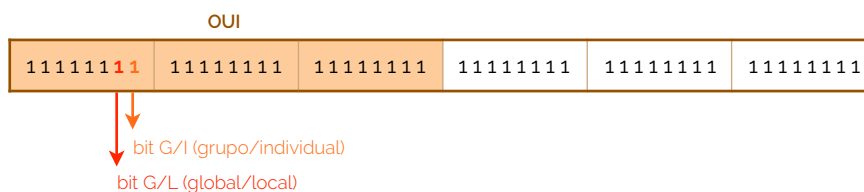


figura 2.2
endereço IEEE

Quando um determinado bloco é atribuído pela IEEE a um fabricante², este terá os últimos três bytes disponíveis, para além do OUI, para definir endereços diferentes para NIC's diferentes. Como podemos ver pela Figura 2.2, no primeiro byte da OUI temos dois bits sinalizados. Acontece que estes dois bits que referimos têm significados especiais: o sétimo bit do primeiro byte da OUI toma o valor '0' se for um endereço globalmente atribuído ou '1' se for administrado localmente (os endereços atribuídos pela IEEE têm sempre o bit a '0'); o oitavo bit do primeiro byte da OUI é '0' se o endereço for um endereço do tipo unicast ou '1' se for multicast (os endereços atribuídos pela IEEE são

² Para ver todos os OUI atribuídos pela IEEE podes consultar o link seguinte: <http://standards.ieee.org/regauth/oui/oui.txt>

sempre dados com este bit a '0'). Já que referimos as diferenças entre tipos de endereços na atribuição do OUI só nos falta referir uma delas, que é o endereço broadcast, por norma IEEE, é todo preenchido a '1', isto é, é sempre o endereço FF-FF-FF-FF-FF-FF.

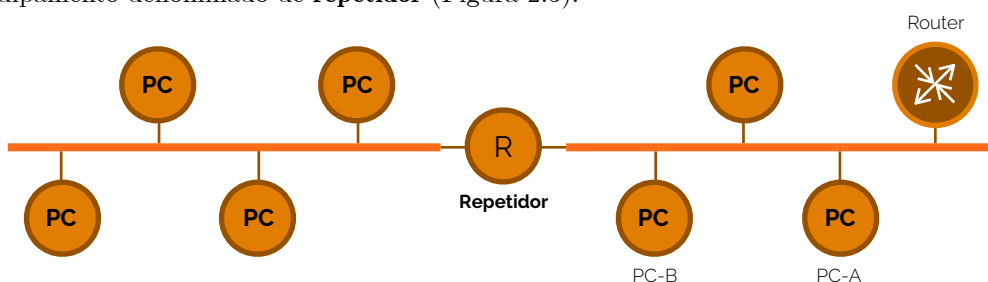
Protocolo Ethernet (IEEE 802.3)

O protocolo **Ethernet** foi inventado em 1973 por Bob Metcalfe e David Boggs, no Centro de Investigação da Xerox, em Palo Alto (*Xerox Palo Alto Research Center*) como forma de efetuar uma partilha de ficheiros entre várias estações. Como base para o acesso ao meio de partilha, os investigadores usavam os endereços MAC atribuídos por um processo CSMA/CD - *Carrier Sense Multiple Address with Collision Detection* (estudaremos mais à frente este conceito).

Neste protocolo, no entanto, ainda havia algumas limitações, embora muitas delas passíveis de serem resolvidas, porque se o número de terminais fosse maior que o número máximo de equipamentos permitidos por ligação partilhada, então ter-se-ia que usar um equipamento denominado de **repetidor** (Figura 2.3).

Ethernet

repetidor
figura 2.3



Este protocolo têm evoluído bastante desde a data da sua criação. Ao longo do tempo foram criados vários standards de utilização como o 10Base5, 10Base2, 10BaseT, neste caso, para ligações com taxas de transferência de 10Mbps.

Consideremos o seguinte cenário, em que o PC-A envia um pacote para a rede e, em simultâneo, o PC-B envia outro pacote. O que é que acontece? Ambos os pacotes chegam aos destinos? Nenhum deles chega ao destino? Para resolver este problema tem de existir um mecanismo que previna situações deste género - o **CSMA/CD**. Este procedimento denominado de *Carrier Sense Multiple Address with Collision Detection* faz com que, antes de um determinado terminal enviar um pacote para a rede, primeiro verifique se a ligação partilhada está a ser utilizada por outro terminal e, mais que isso, verifique se os dados que quer enviar já não se encontram a "navegar" pelo meio. Vejamos então, com mais detalhe, como funciona CSMA/CD.

CSMA/CD

Quando um terminal precisa de enviar um frame para a rede, primeiro este tem de escutar a rede e verificar se mais alguém está a usar o meio partilhado. Se não houver ninguém a usar o meio partilhado, então pode enviar; caso alguém esteja a usar o meio, então ele deve esperar um determinado tempo, aleatório³ (pelo **algoritmo de recuo binário exponencial truncado**), e depois enviar. Este protocolo diz-se **1-persistente** porque as estações estão sempre à escuta enquanto o meio estiver ocupado e, mal ele deixe de o estar, estas transmitem imediatamente.

algoritmo de recuo binário exponencial truncado, 1-persistente

Quando uma estação transmissora deteta uma colisão, esta deve interromper a transmissão da sua trama e enviar um sinal **JAM** (sequência de bits sinalizadora). Depois do envio deste último sinal, a estação, mais uma vez, deve esperar um tempo aleatório até retransmitir.

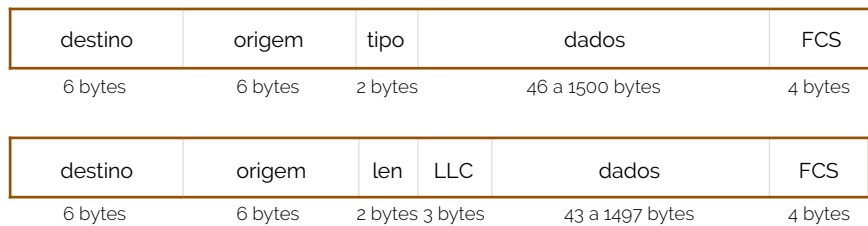
JAM

As **tramas Ethernet** têm dois possíveis tipos: Ethernet II e a 802.3. Em ambos os casos, existem 6 bytes de endereços IEEE, mas dependendo do tipo, existe um campo de tipo, com 2 bytes, onde se define o protocolo transportado no campo de dados, como o protocolo IP, um campo de tamanho, onde se designa o tamanho da trama, um campo

tramas Ethernet

³ O tempo de espera deve ser aleatório e calculado através do algoritmo de recuo binário exponencial truncado, que aumenta o intervalo de aleatoriedade, cada vez que o terminal verifica que não pode transmitir. A duração base da ranhura é de 64 bytes, isto é, o tamanho mínimo de uma trama Ethernet.

LLC, onde se especificam vários campos incluindo os tipos de protocolo e um FCS (*Frame Check Sequence*), usado para a verificação de erros. Na Figura 2.4 podemos verificar ambas as tramas Ethernet II e 802.3.



Ethernet II

802.3

figura 2.4
tramas Ethernet e 802.3

Para garantirmos que todas as estações presentes num meio comum e partilhado detetam colisões, há que fazer com que o tempo de transmissão de uma trama seja maior que o tempo de **round-trip** (ida e volta).

round-trip

Mas como temos dito, as colisões são processos que podem ocorrer num mesmo meio de partilha em comum, isto é, na Figura 2.3, se houver uma colisão no meio à direita do repetidor não significa que haja a mesma colisão no meio à esquerda do repetidor. Por outras palavras, os repetidores definem **domínios de colisão**. Um outro equipamento que define um domínio de colisão é o hub. Por domínio de colisão podemos então definir um secção de uma rede onde frames transmitidos em simultâneo colidem uns com os outros.

domínio de colisão

Então mas qual é a diferença entre repetidores/hubs e switches? Enquanto que os repetidores e os hubs interligam segmentos (meios de partilha) de LANs, os switches (ou **bridges** - geralmente usa-se este nome quando se pretende designar um switch de duas portas) interligam diferentes LANs. Para além desta característica, os switches também permitem funções de salvaguarda e reencaminhamento (cada frame que chega é preservado em memória local e, só depois, é que é reencaminhado para a devida porta) e filtragem (cada frame que chega só pode ser reencaminhada para algumas das portas). Dadas estas características, as portas podem operar a diferentes taxas de transmissão, podendo reencaminhar um frame que chegue para uma só porta caso saiba "onde" é que a estação-destino se encontra. Mais, com isto as colisões deixam de ser um problema (os switches podem receber frames e enviar outros em portas diferentes, em simultâneo) e a largura de banda agregada não é limitada pela taxa de transmissão das portas.

bridges

Para que o switch se comporte como referimos, ele terá de conter alguma forma de manter informações acerca da rede em que se encontra: tem **tabelas de encaminhamento**. As tabelas de encaminhamento dos switches são estruturas de dados que compreendem endereços MAC, interfaces e tempos de vida (TTL), fazendo correspondências entre estes. Mas como é que é feita a manutenção destas tabelas? Consideremos a seguinte rede da Figura 2.5.

tabela de encaminhamento

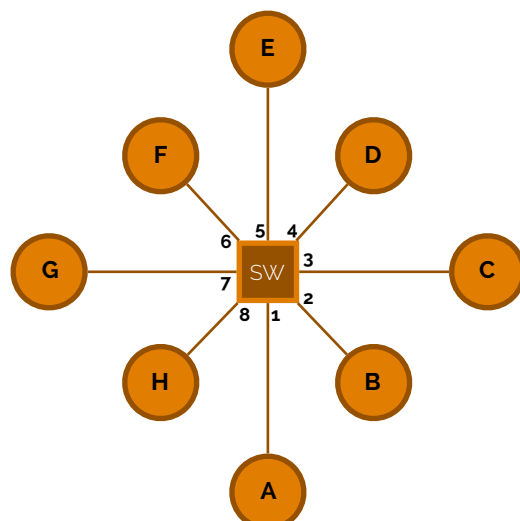


figura 2.5

Conforme a Figura 2.5, consideremos então um switch com 8 interfaces todas elas ligadas a um terminal. Como é que o switch sabe, então, que G é atingido via 7, por exemplo? Até chegar o momento em que o switch sabe responder a essa questão, ele terá de aprender que estações é que podem ser atingidas por cada uma das suas interfaces. Para tal, quando uma trama é recebida numa interface, o switch regista na tabela de encaminhamento uma entrada com o endereço MAC origem da trama e a interface de entrada. Inicialmente a tabela encontra-se vazia, mas mal receba, por exemplo, um pacote vindo de D para G, a tabela fica com os detalhes de D, como se pode verificar na Figura 2.6.

endereço MAC	interface	TTL
D	4	60

figura 2.6

O passo seguinte será reencaminhar o pacote recebido para G. Mas agora há um problema - por que porta é que o switch chega a G? Para tal ele tem de executar o algoritmo de **store & forwarding** (salvaguarda e encaminhamento). Assim, a primeira coisa a fazer é guardar na tabela de encaminhamento a interface do emissor da trama. Depois há que procurar na tabela de encaminhamento uma entrada com o endereço MAC destino. Como não vai encontrar essa entrada na tabela, ele terá de enviar o pacote para todas as interfaces exceto a de entrada - a este processo dá-se o nome de **flooding** (em português, inundação). Caso, porventura, encontrasse na sua tabela o endereço MAC de destino, então ele terá de julgar se o destino é a mesma interface em que a trama foi recebida (descartando-a, caso se verifique essa situação) e, caso contrário, encaminha a trama pela interface indicada - a este processo dá-se o nome de **forwarding** (em português, encaminhamento). Por outras palavras, de forma sumária, se o destino da trama não for conhecido, então ocorre flooding, caso contrário ocorre forwarding. Quando acontece flooding, a estação para a qual o pacote estava destinado envia um reconhecimento do pacote para o emissor, fazendo com que o switch receba mais uma entrada na sua tabela (Figura 2.7).

store & forwarding

flooding

forwarding

endereço MAC	interface	TTL
D	4	60
G	7	60

figura 2.7

Consideremos então, numa escala maior, a rede da Figura 2.8.

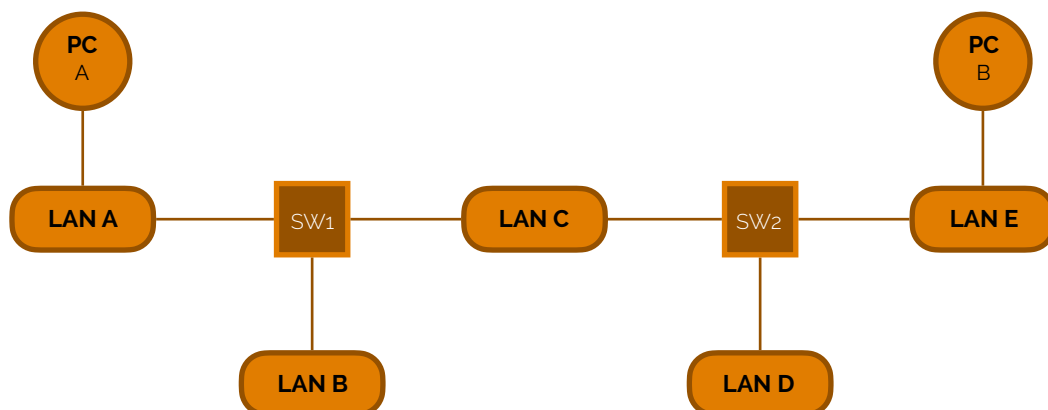


figura 2.8

O que é que acontece numa rede que é composta por vários switches? O processo de auto-aprendizagem e encaminhamento funciona precisamente da mesma forma. Vamos então considerar que o computador pessoal A envia um pacote para o computador pessoal B. Então o pacote sai do PC-A, passa pela LAN A e chega ao Switch 1 - o que é que o switch faz? O Switch 1 vai então adicionar uma entrada na tabela dizendo que, na porta 1

(consideremos que as portas são nomeadas no sentido contra-horário, começando com '1' às 9h) chega-se ao computador pessoal A e, como desconhece o caminho para o computador pessoal B, faz flooding, isto é, envia o mesmo pacote para todas as suas portas, à exceção da porta 1, de onde veio. Ao fazer flooding, a LAN B vai receber um pacote estranho e irá desprezá-lo e a LAN C também irá receber o mesmo pacote, reencaminhando-o para o Switch 2 - o que é que o switch faz? O Switch 2 vai então adicionar uma entrada na tabela dizendo que, na porta 1 chega-se ao computador pessoal A e, como desconhece o caminho para o computador B, faz flooding. Ao fazer flooding, a LAN D vai receber um pacote desconhecido e irá desprezá-lo, e a LAN E irá receber o mesmo pacote, reencaminhando-o para o computador pessoal B, que lhe faz parte. Este processo está sintetizado na Figura 2.9.

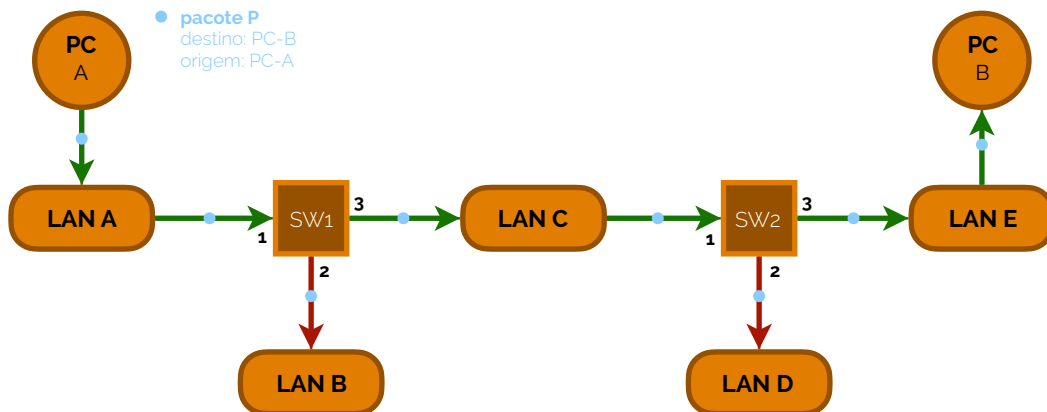


figura 2.9
flooding

Agora, o computador pessoal B pretende enviar um pacote para o computador pessoal A. Então, para isso, o pacote sai do PC-B, passa pela LAN E, e chega ao Switch 2, onde se regista que o PC-B é acessível através da porta 3 e, como já se conhece o caminho para o computador pessoal A, apenas se envia para a porta que lhe corresponde - porta 1. Assim há um forwarding para a porta 1, pelo que o pacote segue viagem para a LAN C até ao Switch 1, este, que regista na sua tabela que o computador pessoal B é-lhe acessível pela porta 3 e, já conhecendo o caminho para o computador pessoal A, envia o pacote para a porta 1, passando pela LAN A e, finalmente, chegando ao computador pessoal A. Este processo está sumariamente representado na Figura 2.10.

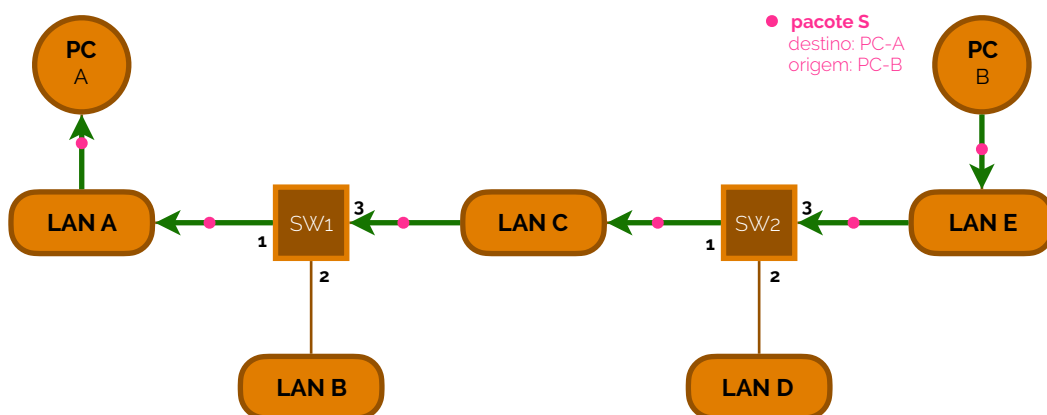


figura 2.10

Mas como já referimos, as tabelas de encaminhamento dos switches não possuem apenas uma correspondência entre um endereço MAC e uma interface - também possuem um campo que especifica um tempo de vida (TTL). Este valor é predefinido cada vez que o switch recebe um pacote da estação terminal de origem. Entretanto, esta entrada na tabela é eliminada quando, passados TTL segundos, não forem recebidos quaisquer pacotes de tal origem - diz-se então que a entrada **expirou**. Assim, se mudarmos a posição do computador pessoal B, da LAN E para a LAN D, por exemplo (seguindo o exemplo da

expirou

Figura 2.9), se enviarmos o mesmo pacote de PC-A para PC-B, este será desperdiçado (perdido), dado que a sua entrada ainda não expirou no Switch 2 e o PC-B ainda não deu a conhecer ao Switch 2 que está na LAN D (Figura 2.11).

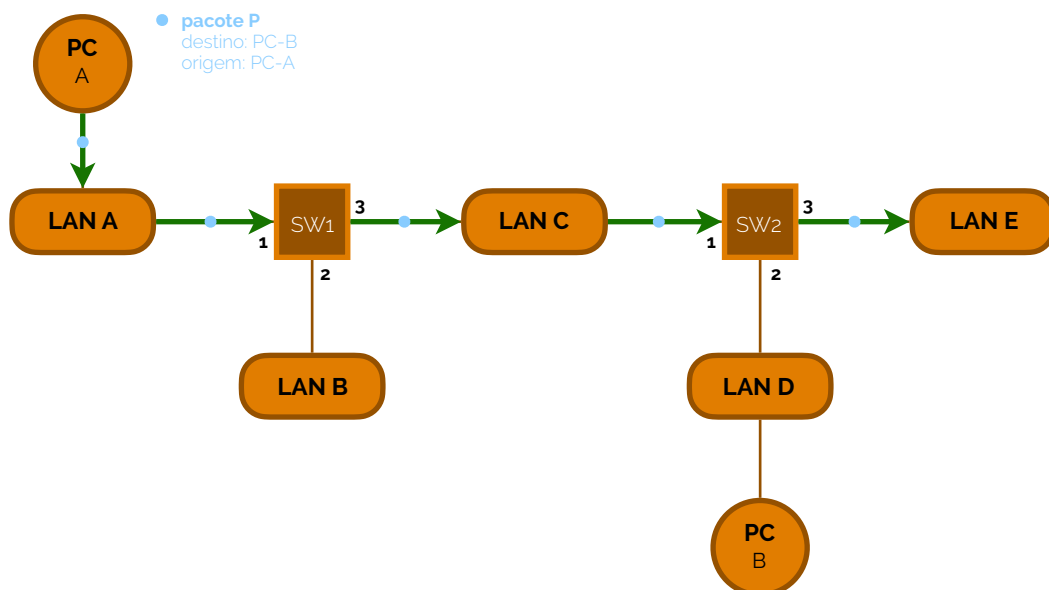


figura 2.11

Se fizermos a interligação de redes de área local podemos ter domínios de colisão diferentes, mas também podemos ter diferentes **domínios de broadcast**, caso interliguemos redes com routers - aqui cada rede terá um endereço IP diferente. Basicamente, diferentes tipos de equipamento definem diferentes tipos de domínio de rede. Os terminais que se encontram separados por hubs ou repetidores estão no mesmo domínio de colisão e broadcast. Um switch (ou bridge) separa uma rede em diferentes domínios de colisão, mas define um mesmo domínio de broadcast. Um domínio de broadcast é assim uma secção de rede onde todas as estações comunicam diretamente. Já um router separa a rede em diferentes domínios de broadcast, atribuindo a cada domínio de broadcast um endereço IP próprio, pelo que estações em diferentes domínios de broadcast não podem comunicar diretamente (terão de comunicar via IP de default gateway).

domínios de broadcast

Consideremos então a rede da Figura 2.12, onde teremos de avaliar quais são os domínios de colisão (quantos e quais são) e quais são os domínios de broadcast (quantos e quais são).

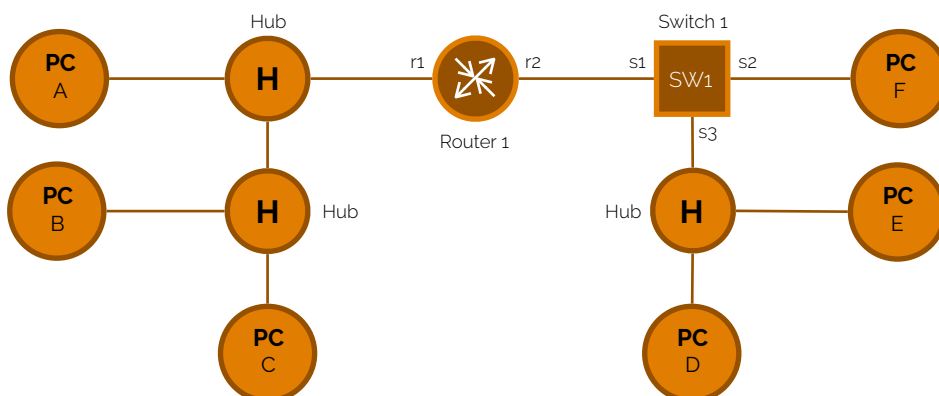


figura 2.12

Primeiro, por ser mais fácil, podemos começar por avaliar quais e quantos são os domínios de broadcast. Para isso, há que saber que equipamento é que divide domínios de broadcast - o router. Assim sendo, basta verificar quantas interfaces tem o router ligadas, que o número de interfaces será o número de domínios de broadcast da Figura 2.12 - duas interfaces, logo dois domínios (o primeiro domínio é constituído por PC-A, PC-B, PC-C e r1 e o segundo domínio é constituído por PC-D, PC-E, PC-F e r2).

A segunda parte do exercício pede-nos que descrevamos quais e quantos são os domínios de colisão. Os equipamentos que criam diferentes domínios de colisão são os switches, em cada rede. Assim sendo, na rede à esquerda do router teremos um único domínio de colisão, dado que todos os equipamentos lá presentes partilham a mesma ligação e, do lado direito temos um switch, que cria um domínio de colisão nos equipamentos PC-D, PC-E e *s3* (o domínio *s2* e PC-F não é considerado domínio de colisão porque não há colisões nesta ligação, dado que só existe um terminal no meio de partilha). Na Figura 2.13 está uma representação dos domínios de broadcast (a amarelo) e de colisão (a verde).

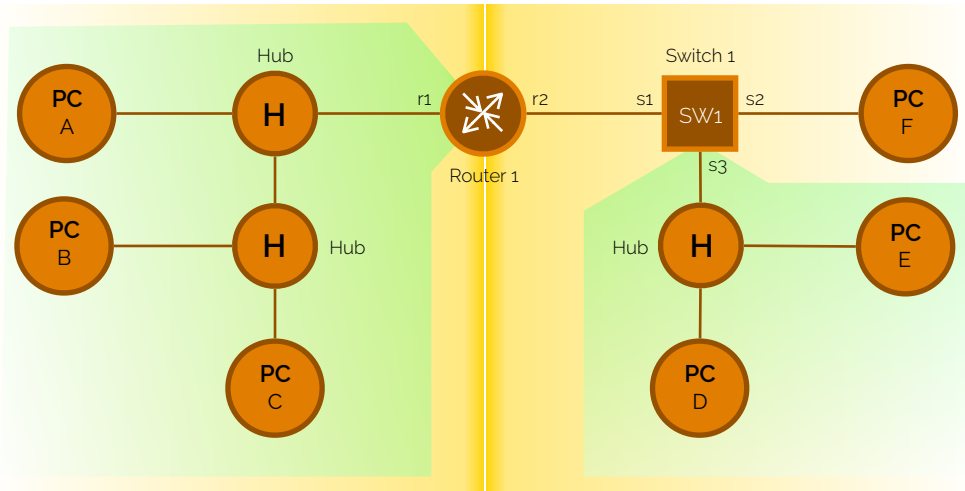


figura 2.13
domínios de colisão e
broadcast

Então mas qual é a diferença entre routers e switches? Quando falamos em rede falamos, implicitamente, numa organização em camadas (que iremos estudar mais à frente em detalhe). A camada mais baixa é a camada física (correspondente a todo o hardware que compõe uma rede). De seguida vêm outras camadas, por ordem, ligação, rede, transporte e aplicações. Os routers trabalham com as três primeiras camadas, mas os switches só trabalham com as primeiras duas camadas. Por conseguinte, estes não têm, por exemplo, endereço IP e os seus encaminhamentos serão feitos sempre a nível 2 (nível de ligação de dados). Existem outros switches (denominados Switches de camada 3, ou simplesmente **switch L3**) que trabalham também na camada de rede.

switches L3

Em suma, na lista seguinte estão as funções principais de cada um dos equipamentos que estudámos até agora:

- **repetidor/hub:** operando no nível físico (camada 1), regenera os sinais (considera-se como um hub, um repetidor com múltiplas portas);
- **bridge/switch:** sendo do tipo *store & forward* opera ao nível da camada da ligação (camada 2) e serve para interligar dois ou mais domínios de colisão, pelo que comuta com base nos endereços MAC (considera-se como um switch, um bridge com múltiplas portas);
- **router:** sendo do tipo *store & forward* opera ao nível de rede (camada 3) e comuta com base nos endereços de nível 3, como é exemplo o endereço IP.

repetidor/hub

bridge/switch

router

3. Redes de Área Locais Virtuais (VLAN)

Como já devemos saber, uma LAN (rede de área local) define um domínio de broadcast e todos os terminais a si ligados podem comunicar de forma direta. Mas será que podemos modificar isso, isto é, subdividir o domínio de broadcast na nossa VLAN, mantendo, em termos físicos, a mesma rede? Através do conceito que pretendemos introduzir agora sim - eis as **VLAN's**. Uma VLAN é então uma rede de área local definida em termos lógicos, não físicos.

VLAN

Configuração de VLAN em switches

Quando um switch é configurado com diferentes VLAN's, este atua como se fosse um switch em separado para cada uma das VLAN's. Na Figura 3.1 podemos ver um switch que foi configurado de forma a que as portas 1 a 5 fossem pertencentes à VLAN 1 e as portas de 7 a 12 fossem pertencentes à VLAN 2. Assim os terminais ligados à VLAN 1 podem comunicar diretamente entre eles e os da VLAN 2 também podem comunicar diretamente entre eles. Mas será que é possível que um terminal da VLAN 1 comunique diretamente com um equipamento na VLAN 2? Não - essa é a utilidade das VLAN's. Para termos comunicações globais precisamos de um router, e de forma a que funcione, os endereços IP de cada VLAN terão de ser diferentes, como na Figura 3.2.

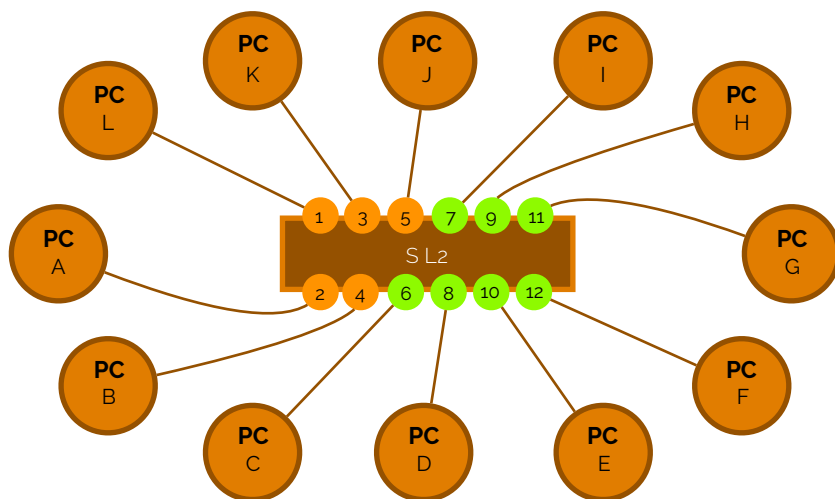


figura 3.1

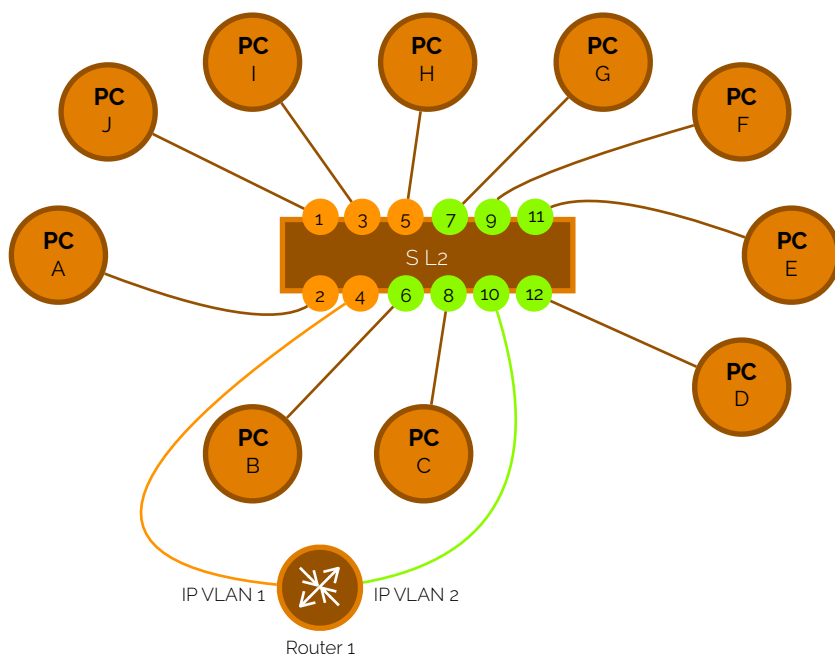


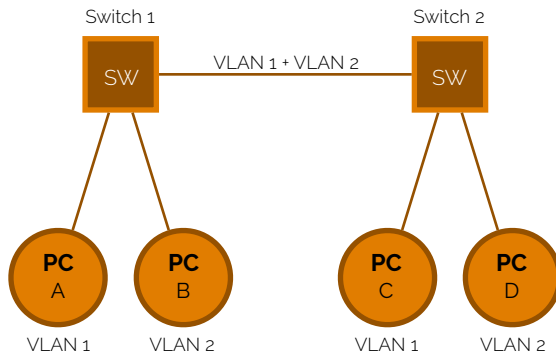
figura 3.2

As VLAN's, no entanto, não precisam de um router para se poderem interligar. Existem duas formas para efetuar comunicações entre duas VLAN distintas: uma possível solução é aplicar o que está representado na Figura 3.2, isto é, interligar duas VLAN com um router; uma segunda solução é criar uma ligação **interswitch**, que, na mesma ligação, possua duas VLAN diferentes. Mas isto leva-nos a uma questão pertinente: como é que distinguimos dois pacotes de duas VLAN diferentes, na mesma ligação? A IEEE definiu

interswitch

assim um protocolo específico para a manipulação de VLAN, denominado de **802.1Q**, que acaba por permitir VLAN's entre switches de diferentes fabricantes, inclusivé.

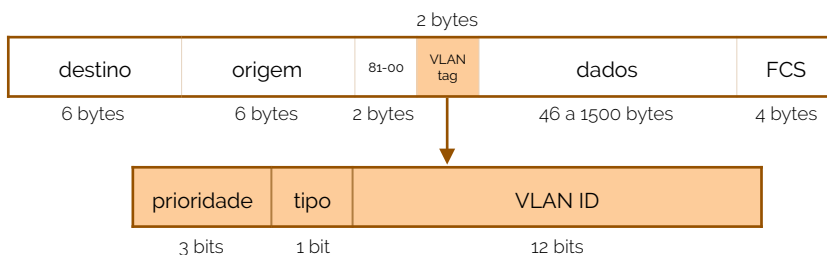
Consideremos assim o caso em que temos VLAN's definidas em switches diferentes, como podemos ver na Figura 3.3. Usando portas interswitch (denominadas como *tagged ports*) podemos, através do protocolo 802.1Q manter as mesmas configurações das VLAN's. Claro está que, as tramas usadas sob a regência deste protocolo terão ligeiras diferenças - passarão a ter uma **etiqueta VLAN**.



etiqueta VLAN

figura 3.3
switches e portas interswitch

Com o protocolo 802.1Q, como já foi referido, as tramas terão uma forma ligeiramente diferente, pois desta irá agora constar também uma etiqueta VLAN. As etiquetas VLAN são 4 bytes que se irão localizar entre o campo de endereço de origem e o campo de tipo. Os primeiros 2 bytes têm sempre o valor de 0x8100 e os outros dois bytes incluem um **VLAN ID** (identificador da VLAN), com 12 bits. Estes bytes podem ser visto como uma etiqueta que é inserida em cada frame. Na Figura 3.4 podemos ver a estrutura de uma trama de um pacote Ethernet com etiqueta VLAN (802.1Q).



802.1Q

figura 3.4
trama 802.1Q

Na receção de um pacote vindo de uma porta interswitch por um switch, este só poderá aceitar os que cujos frames tenham o campo 0x8100.

Então mas afinal qual é a vantagem de ter VLAN's? Nas LAN's tradicionais os terminais ligados à mesma rede de área local estão limitados geograficamente pela extensão máxima da tecnologia. Pelo contrário, com VLAN's, essa limitação deixa de existir e o administrador de rede pode agrupar os terminais nas mesmas VLAN's segundo outros critérios que permitam uma melhor gestão da rede. Outra razão é o broadcast, que aumenta exponencialmente com o número de estações numa LAN, embora, numa VLAN, seja possível segmentar a rede por forma a manter o tráfego de broadcast num valor razoável.

Como acabámos de referir, as VLAN's permitem a segmentação da rede na camada protocolar de ligação (*link layer*) que anteriormente só era possível através de routers na camada protocolar de rede (*network layer*).

4. Spanning Tree Protocol (STP)

Uma rede de telecomunicações é um conjunto de ligações ponto-a-ponto entre equipamentos - matematicamente podemos traduzir esta estrutura como um **grafo**, onde cada nó representa um equipamento da rede e as arestas representam as ligações. No nosso caso, cada aresta terá um **custo** associado.

grafo

custo

Estrutura de dados (árvore abrangente)

Consideremos o grafo da Figura 4.1. Neste grafo temos seis nós (equipamentos) ligados entre eles por ligações com um determinado custo.

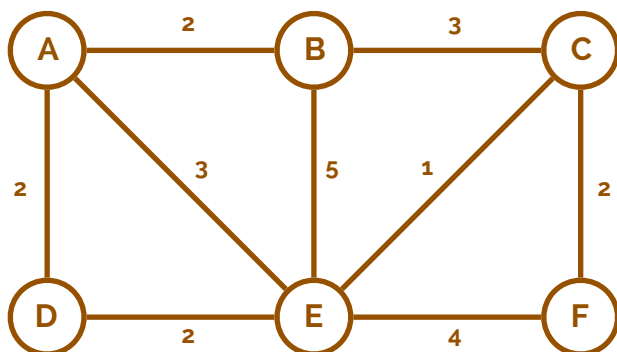


figura 4.1

Em Matemática Discreta (als2) discutimos os conceitos de árvore e de árvore abrangente. Então, revendo, uma **árvore** é um subconjunto das arestas de um grafo que não contém ciclos (define apenas um caminho por par de nós). Já uma **árvore abrangente** é uma árvore que liga todos os nós, sem exceção, definindo um só caminho por cada par de nós. Na Figura 4.2 está um exemplo de árvore abrangente.

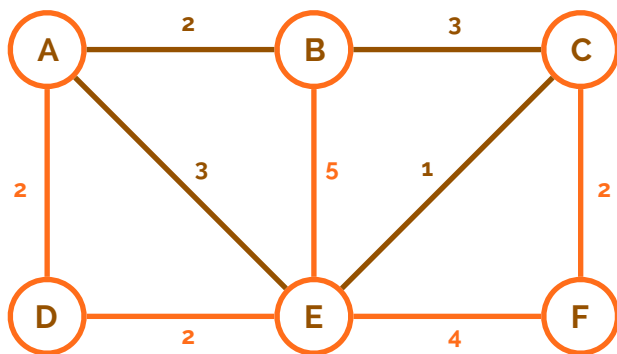


figura 4.2

Uma forma de determinar uma árvore abrangente - em inglês **spanning tree** - é computando uma spanning tree de custo mínimo. Uma **árvore abrangente de custo mínimo** é uma árvore cuja soma de todos os custos das suas arestas é mínimo. A Figura 4.3 mostra, assim, uma árvore abrangente de custo mínimo com custo 9. Se verificarmos bem, podemos reparar que não existe outra árvore com custo inferior a 9 - embora, num caso geral, possam haver múltiplas árvores com custo mínimo.

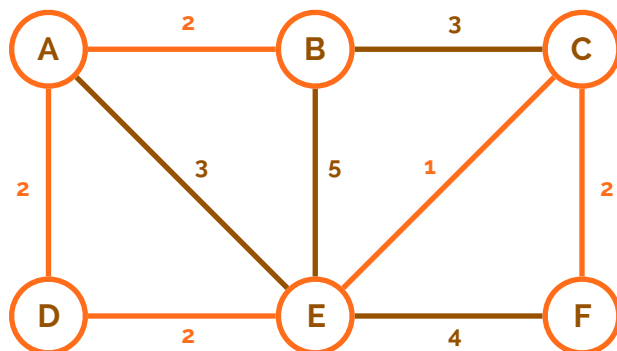


figura 4.3

Mas como é que podemos calcular o custo mínimo e a respetiva árvore abrangente? Consideremos então, de volta à Figura 4.1, o nó A como **nó raiz**. Se verificarmos todos os caminhos de todos os nós para A, podemos verificar que o caminho

árvore
árvore abrangente

spanning tree
árvore abrangente de custo mínimo

nó raiz

mais curto: desde B é {B, A}; desde C é {C, E} e {E, A}; desde D é {D, A}; desde E é {E, A}; e desde F é {F, C}, {C, E} e {E, A}. As arestas incluídas pelo menos uma vez nestes caminhos definem a spanning tree dada pelas ligações {A, B}, {A, D}, {A, E}, {C, E} e {C, F} - note-se que por este método não se obtém as árvores abrangentes de custo mínimo, mas antes o custo mínimo para cada ponto, à vez.

Ligação de árvores abrangentes em LANs

Quando ligamos diferentes switches é desejável que tenhamos uma ligação redundante, isto é, um conjunto de ligações que possam manter uma rede num todo ligada quando algumas ligações falham. Na Figura 4.4 podemos ver um exemplo onde se tenta representar uma rede composta por 3 switches, cada um num edifício diferente. Caso uma das três ligações efetuadas falhe, haverá sempre uma que consiga suportar a sua ausência.

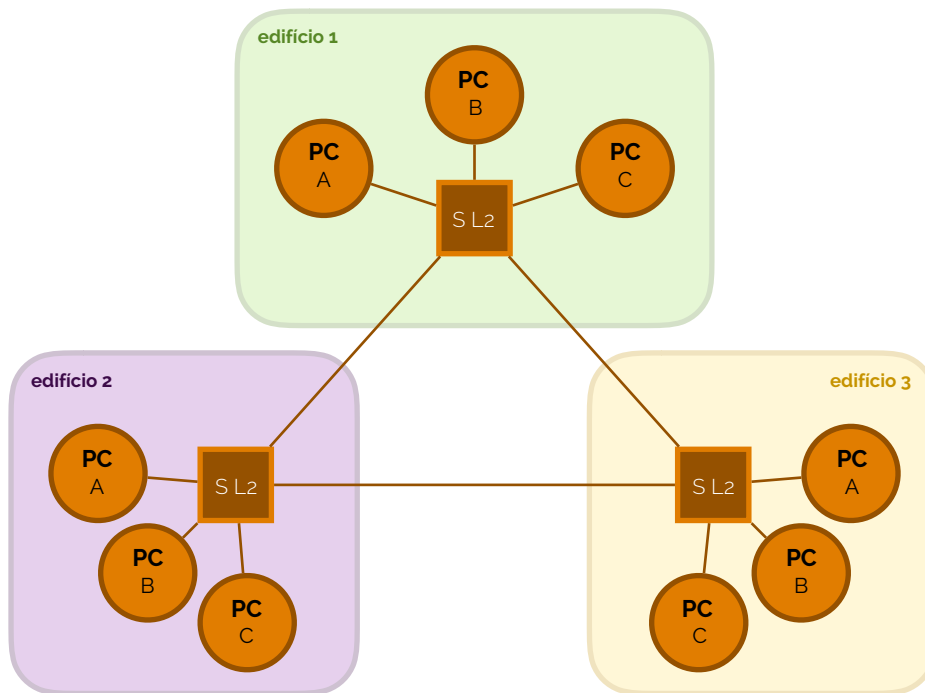


figura 4.4

No entanto, num nível lógico, existem ciclos de roteamento, o que irá causar colapso da rede, dado que frames que possam ter intenção de broadcast circularão na rede em ciclos intermináveis. Para resolver este problema há um protocolo que é usado entre switches e que lhes permite decidir que conexões devem encaminhar frames e que conexões devem suspender a propagação destes. Este protocolo, designado por **IEEE 802.1D**, é vulgarmente denominado de **Spanning Tree Protocol (STP)**.

Basicamente, o que é feito no STP é aplicar os conhecimentos que retemos acerca das árvores abrangentes em Matemática Discreta (als2), em termos de redes, mais precisamente na manutenção pelos switches. Para calcular o percurso mais curto são usadas as **equações de Bellman**, onde, considerando um grafo a cujas arestas é atribuído um custo não negativo e existe um nó raiz, o custo do percurso mínimo de um nó para a raiz é a soma do custo do arco que une esse nó ao nó que lhe segue no percurso mínimo com o custo do percurso mínimo desse nó para o nó raiz, e o **algoritmo de Bellman-Ford** (distribuído e assíncrono). Este algoritmo pode ser executado de uma forma distribuída entre todos os nós onde cada nó resolve as equações de Bellman baseadas na informação dos seus nós vizinhos, e a troca de informação entre os nós pode ser assíncrona.

A cada nó i , o algoritmo funciona da seguinte forma: o nó i envia periodicamente a sua estimativa D_i para os vizinhos do seu custo mínimo em caminho para a raiz; quando recebe as estimativas dos seus vizinhos, este atualiza o seu custo com $D_i = \min(d_{ij} + D_j)$

IEEE 802.1D
Spanning Tree Protocol
 < [IEEE 801.D](#)

equações de Bellman
 • **Richard Bellman**

algoritmo de Bellman-Ford
 • **Lester Ford Jr.**

entre todos os seus vizinhos j . O vizinho j que $d_{ij} + D_j$ é mínimo torna-se o nó seguinte num caminho para a raiz.

Na Figura 4.5 o nó D recebeu dos seus vizinhos os valores estimados de custo 3 (de B), 7 (de C), 5 (de E) e 2 (de F). Por cada vizinho, agora, D soma a estimativa vizinha com a sua, o que dá um custo de: $6 + 3 = 9$ para B; $3 + 7 = 10$ para C; $5 + 1 = 6$ para E; e $5 + 2 = 7$ para F. O valor mínimo é 6 através de E, pelo que a estimativa atual passa a ser 6 e E passa a ser o nó seguinte no caminho para a raiz.

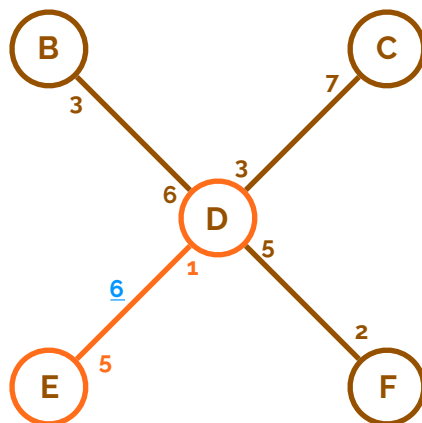


figura 4.5

Então como é que se processa o Spanning Tree Protocol? Como é que os switches conseguem arranjar o melhor caminho? O algoritmo é o seguinte:

- é escolhido um switch como nó raiz;
- os outros switches usam o algoritmo de Bellman-Ford assíncrono e distribuído para calcular o vizinho no percurso de custo mínimo para o nó raiz;
- as ligações compostas pelos percursos de custo mínimo (de todos os outros switches para a raiz) definem uma árvore abrangente (spanning tree);
- as portas ativas são as das ligações que compõem a árvore abrangente.

Cada switch tem um **bridge ID** (valor identificador do switch). Este valor é um endereço que contém 2 bytes de prioridade (configurável pelo gestor da rede), juntamente com 6 bytes, fixos, sendo este um dos endereços MAC das portas do switch ou qualquer outro endereço único de 48 bits. Na Figura 4.6 temos uma representação deste endereço.

bridge ID



Bridge ID

figura 4.6
bridge ID

Para definir qual é o switch que vai ser a raiz, conceptualmente será sempre aquele que terá o bridge ID mais pequeno. No desenvolver da spanning tree iremos também usar nomenclaturas como bridge designada, porta designada ou porta raiz: uma **bridge designada** é um switch que, numa LAN, é responsável pelo envio de pacotes da LAN para a raiz e vice-versa - a bridge raiz é bridge designada em todas as LANs a que está ligada; uma **porta designada** é uma porta que, numa LAN, é responsável pelo envio de pacotes da LAN para a raiz e vice-versa (é uma das portas da bridge designada); uma **porta raiz** é uma porta que é responsável pela receção e transmissão de pacotes de ou para a raiz. Cada bridge tem associado um custo do percurso para a raiz (**root path cost**) igual à soma dos custos das portas que recebem pacotes enviados para a raiz (porta raiz) no percurso de menor custo para a bridge. Vejamos então o exemplo de rede da Figura 4.7 e tentemos construir a árvore abrangente.

bridge designada

porta designada
porta raiz

root path cost

A primeira coisa a fazer é tentar identificar qual das bridges da nossa rede é que é a bridge raiz: para tal precisamos de verificar qual é que tem o seu bridge ID menor. Como a bridge com bridge ID menor é a que está no topo da figura, com valor de 12, consideraremos esta como bridge raiz.

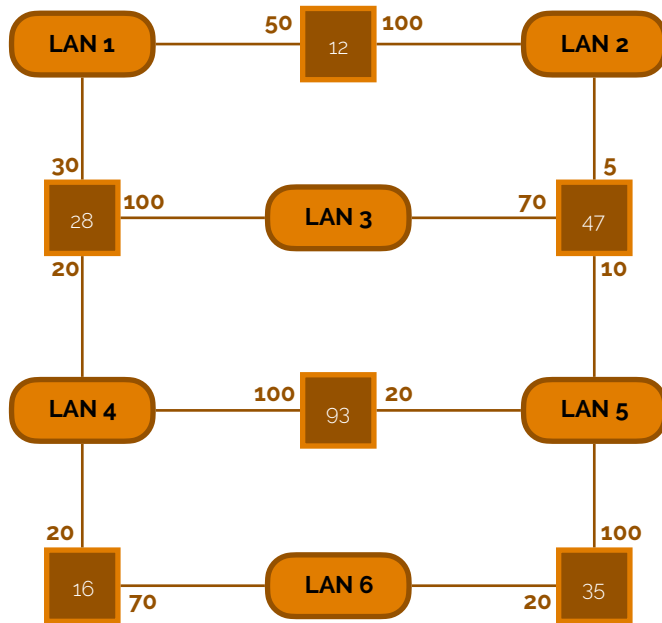


figura 4.7

Estando agora sinalizado - o switch com ID 12 - como switch raiz, uma boa escolha, é tentar resolver quais são as portas raiz - por outras palavras visitamos cada switch e tentamos resolver qual o melhor caminho para a raiz, identificando a porta com essa distinção como porta raiz. A Figura 4.8 mostra a resolução até esse ponto, destacando as portas raiz a verde.

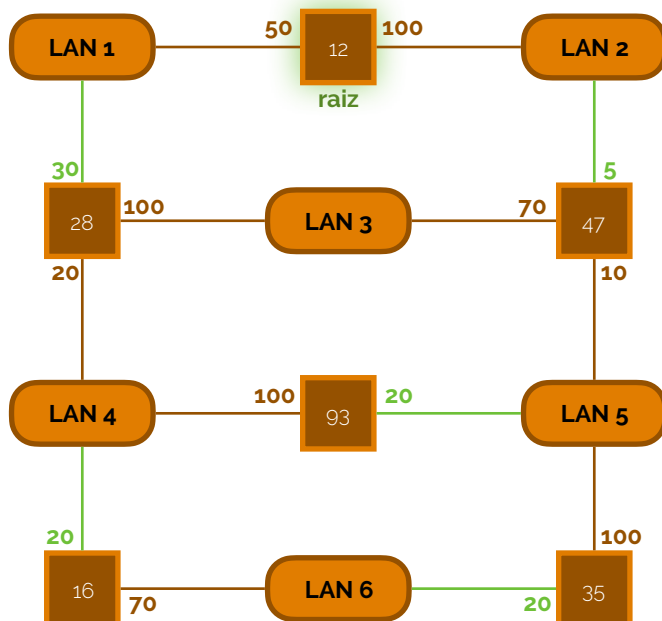


figura 4.8

Tendo as portas raiz já todas determinadas, agora há que determinar quais são as portas designadas. Ora, as portas designadas são as portas que, vindas das LANs, atingem os switches encaminhando os pacotes no caminho para a raiz. Assim, para resolver este problema, precisamos de visitar cada LAN e verificar qual o melhor caminho para a raiz. Uma representação deste passo é visível na Figura 4.9, onde as portas designadas estão indicadas a azul e as portas verdes identificam as portas raiz.

Neste momento, já temos as portas raiz definidas e as portas designadas de cada LAN. A única coisa que falta, é mesmo tornar as outras ligações suspendidas, pelo que existindo fisicamente, elas vão deixar de funcionar, dado que estas não fazem parte do leque de soluções ótimas de ligação entre os vários equipamentos na nossa rede com o protocolo STP. Este procedimento é visível na Figura 4.10.

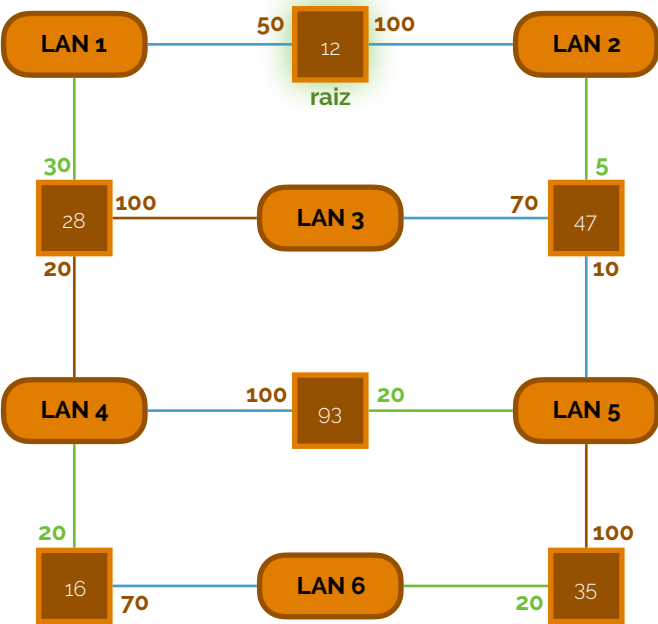


figura 4.9

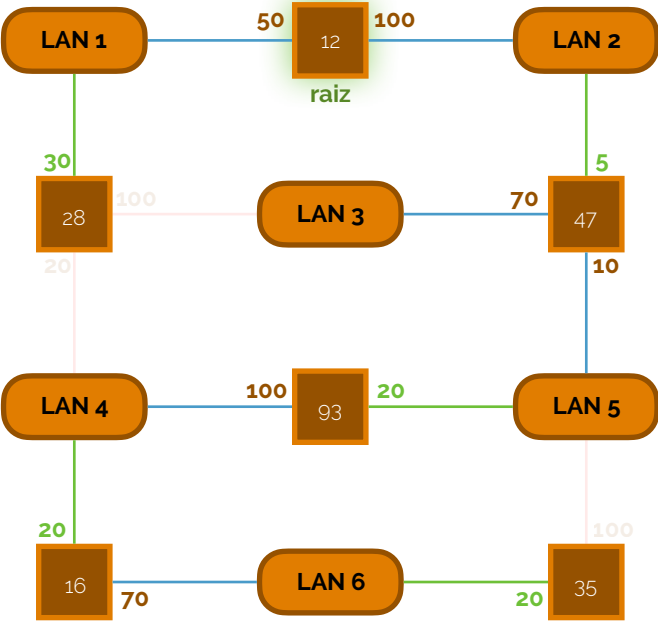
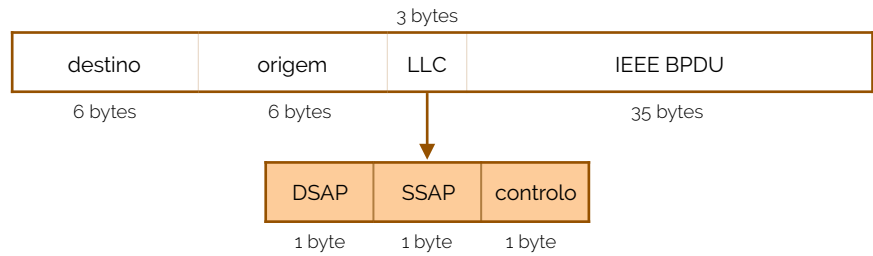


figura 4.10

Mas como é que as spanning trees, em termos reais e lógicos, se constroem e se mantêm? Para construir e manter a spanning tree os switches trocam mensagens especiais entre si, designadas por **BPDU**s (sigla inglesa para *Bridge Protocol Data Units*). Existem dois tipos de mensagens BPDU: mensagem de configuração e a mensagem de *topology change notification* (notificação de mudança da topologia). Basicamente, as tramas Ethernet, para este caso, terão de conter outro tipo de protocolo - o BPDU. Na Figura 4.11 temos então o caso de uma trama 802.3 (Ethernet) com o protocolo BPDU instalado.

BPDU



802.3 (BPDU)

figura 4.11
BPDU

Na trama 802.3 (com BPDU) existem, no lugar do campo LLC, três campos - cada um com 1 byte - *DSAP* (*Destination Service Access Point*), *SSAP* (*Source Service Access Point*) e *control* (controle). Ambos os campos de *DSAP* e *SSAP* têm o valor de 0x42 com mensagens do tipo BPDU.

Já dentro do campo IEEE BPDU temos algumas informações relevantes, como o *Root ID* (identificação do switch raiz), o *Root Path Cost* (custo para a raiz), o campo *Bridge ID* (identificação do switch atual) e *Port ID* (identificação da porta que envia a mensagem de configuração).

Para fazer a manutenção da spanning tree a bridge raiz tem de enviar periodicamente mensagens BPDU de configuração através das suas portas designadas (a periodicidade é definida pelo **hello time** e o seu valor recomendado é de 2 segundos) e para cada mensagem recebida numa porta raiz, cada switch deve calcular a sua própria mensagem de configuração e enviá-la através das suas portas designadas. Consideremos assim a Figura 4.12, onde a bridge raiz (bridge com ID 15 - do topo) envia periodicamente em todas as suas portas designadas uma mensagem de configuração BPDU com o valor da bridge raiz (15), custo do caminho para a raiz (0) e bridge ID (15) - de forma sumária, o que vai enviar será 15.0.15 (na forma de *root.cost.bridgeID*).

hello time

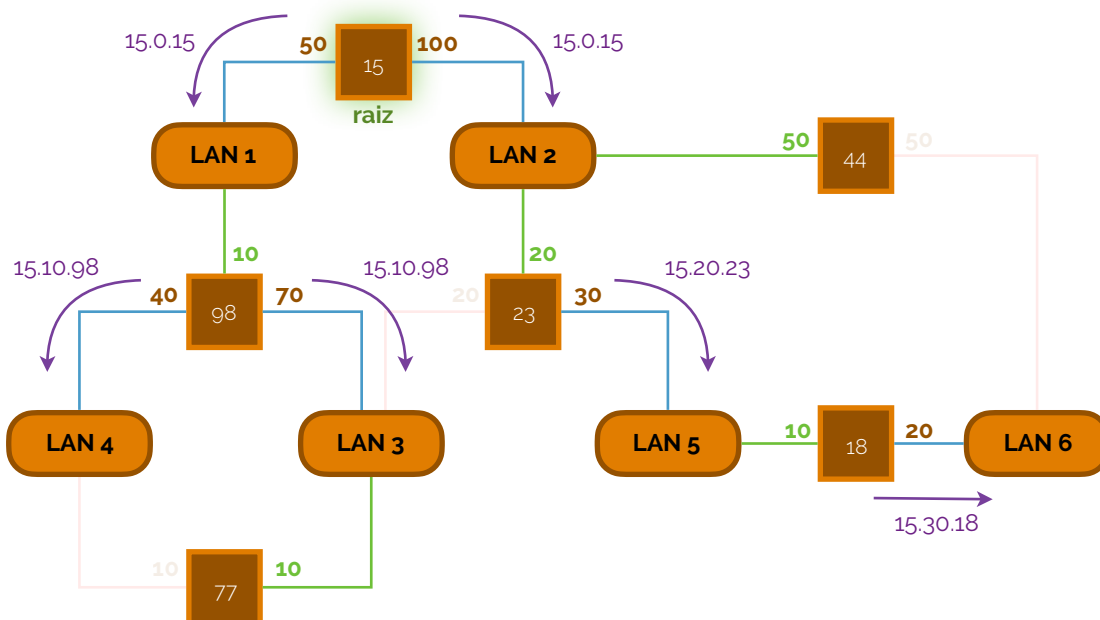


figura 4.12

Para cada mensagem de configuração BPDU recebida pela bridge com bridge ID 98 na sua porta raiz, esta irá enviar para todas as suas portas designadas uma nova mensagem de configuração BPDU com o valor da raiz (15), custo para a raiz (10) e bridge ID (98). E assim sucessivamente.

Mas as mensagens de configuração têm de estabelecer uma ordem, por importância, dado que pode haver alterações que tenham de surgir com alguma urgência. Assim, uma mensagem de configuração c_1 diz-se melhor que uma outra mensagem c_2 se:

- o *root ID* de c_1 for inferior ao de c_2 ;
- sendo os *root ID* idênticos, o *root path cost* de c_1 for inferior ao de c_2 ;
- sendo idênticos os *root ID* e o *root path cost*, o *bridge ID* de c_1 for inferior ao de c_2 ;
- sendo idênticos o *root ID*, o *root path cost* e o *bridge ID*, o *port ID* de c_1 for inferior ao de c_2 .

Assim, na construção de uma spanning tree cada bridge assume inicialmente que é a bridge raiz, fazendo *root path cost* = 0, e envia mensagens de configuração em todas as suas portas. Este processo está ilustrado na Figura 4.13.

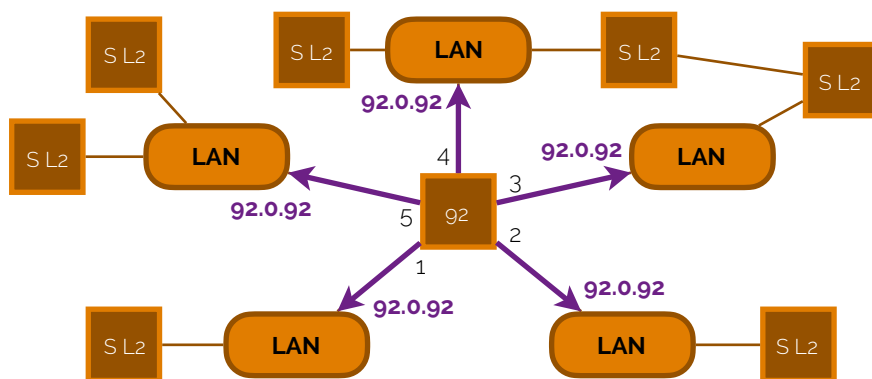


figura 4.13

A qualquer instante, o switch assume que: o root ID é o menor valor de root ID das mensagens de configuração BPDUs que recebe dos seus vizinhos, o seu custo para a raiz é dado pelas equações de Bellman e a sua porta raiz é a porta que liga ao vizinho que forneça o menor custo para a raiz. Neste momento algo como o representado na Figura 4.14 acontece.

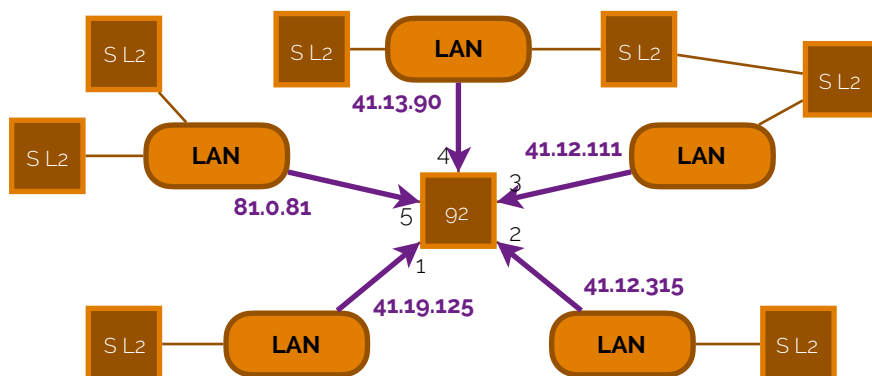


figura 4.14

Assim, no estado da Figura 4.14 a bridge 92 faz estimativas e, assumindo que os custos das portas são unitários cria uma mensagem de configuração BPDUs com a informação 41.13.92, enviando esta mensagem para quem ainda necessita dela, isto é, para aqueles cuja informação é maior do que a que está para ser enviada. Em simultâneo, o switch já pode registrar portas que estarão inativas e porta raiz, como podemos ver na Figura 4.15.

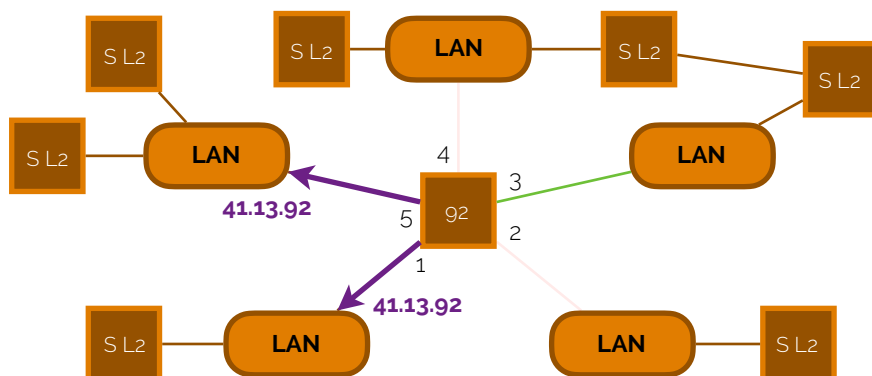
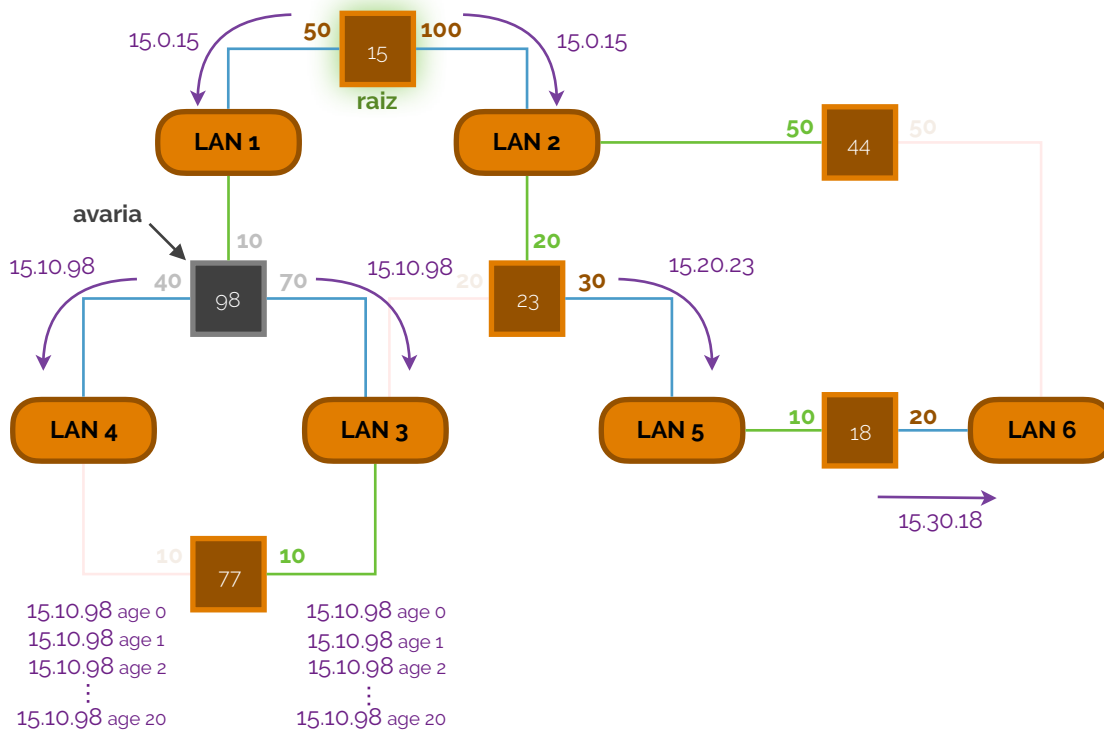


figura 4.15

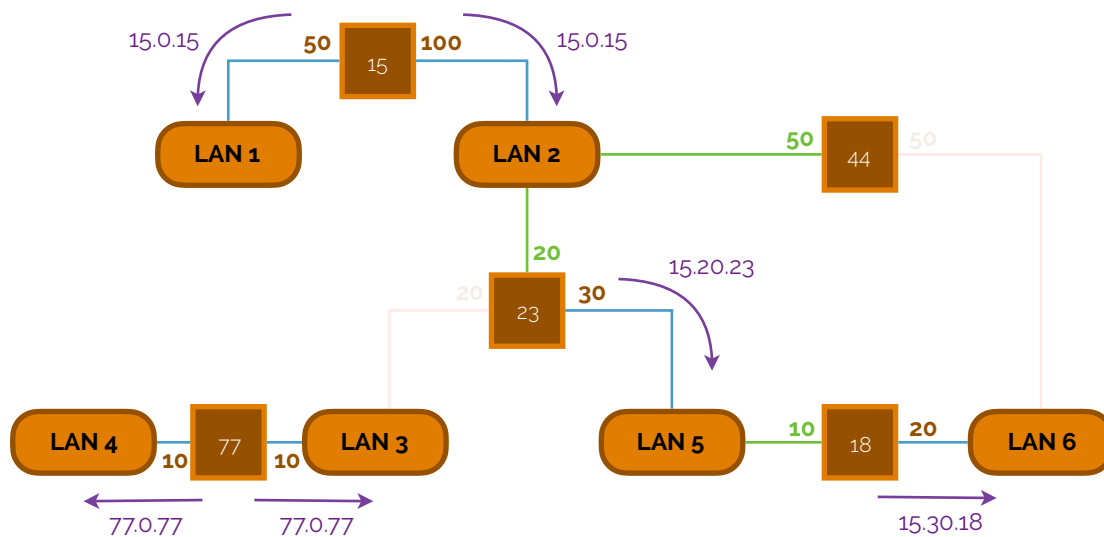
Se acontecer alguma avaria na rede, há sempre uma outra bridge que deixará de receber as mensagens de configuração pela bridge raiz. Nesta situação, a idade da informação recebida na última mensagem de configuração deve atingir o número marcado no campo *max age* (que por definição tem como valor recomendado 20 segundos) e a informação deve ser descartada. No exemplo da Figura 4.16, uma bridge falhou e a bridge 77 deixou de receber mensagens de configuração BPDUs em ambas as suas portas.

figura 4.16



Neste caso, o que vai acontecer é que o processo de criação da rede recomeça e a bridge deverá reconhecer-se como bridge raiz, enviando uma mensagem de configuração BPDU 77.0.77. Esta situação está representada na Figura 4.17.

figura 4.17



Como a bridge 23 tem uma mensagem melhor (15.20.23) que a mensagem recebida (77.0.77), então, numa próxima vez que receba uma mensagem de configuração BPDU na sua porta raiz, esta enviará a sua mensagem de configuração BPDU e ativará a porta de forma a voltar a ligar toda a rede. A mensagem de configuração BPDU com conteúdo 15.20.23 ativa a bridge com valor de identificação 77 para selecionar de forma apropriada a sua própria porta raiz e designar as outras como portas designadas. Este procedimento pode ser verificado na Figura 4.18.

Após a alteração da **topologia da rede** pode existir perda temporária de conectividade se uma porta que estava inativa na topologia antiga ainda não se apercebeu que deverá estar ligada na nova topologia. Na mesma situação também podem existir ciclos temporários se uma porta que estava ativa na topologia antiga ainda não se apercebeu que deverá estar inativa na nova topologia.

topologia da rede

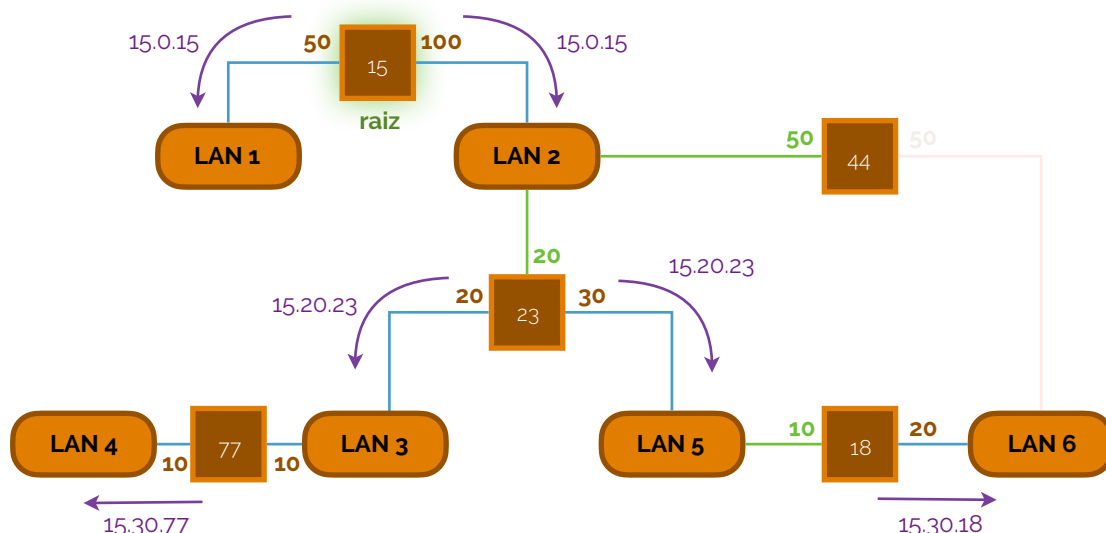


figura 4.18

Para minimizar a probabilidade de se formarem ciclos temporários os switches são obrigados a esperar algum tempo antes de permitirem que uma das suas portas passe do estado inativo para o estado ativo (o tempo de espera é função do parâmetro *forward delay*).

De forma mais sumariada, podemos então dizer que as portas numa spanning tree podem ter **estados**. Entre eles, uma porta pode estar:

- estado **blocking**: os processos de aprendizagem e de expedição de pacotes estão inibidos, pelo que recebe e processa mensagens de configuração;
- estado **listening**: os processos de aprendizagem e de expedição de pacotes estão inibidos, pelo que transita para o estado learning após um tempo de permanência neste estado igual a *forward delay* e recebe e processa mensagens de configuração;
- estado **learning**: os processos de aprendizagem estão ativos, mas o processo de expedição de pacotes está inibido, transitando para o estado forwarding após um tempo de permanência igual a *forward delay* e recebe e processa mensagens de configuração;
- estado **forwarding**: é o estado ativo, onde tanto o processo de aprendizagem como o processo de expedição de pacotes estão ativos, recebendo e processando mensagens de configuração;
- estado **disabled**: estado em que o processo de aprendizagem e de expedição de pacotes estão ambos inibidos e não se participa no processo de spanning tree.

estados

blocking

listening

learning

forwarding

disabled

As tabelas de encaminhamento de cada switch também tem um tempo de vida. Este tempo de vida, se for demasiado longo, pode fazer com que haja um elevado número de pacotes perdidos quando a estação muda de posição. Por outro lado, se este tempo de vida for demasiado curto, então o tráfego na rede pode ser exagerado devido ao processo de flooding. Para tal existem dois tempos de vida: um **tempo de vida longo**, o qual é usado por defeito (cujo valor recomendado é de 5 minutos) e um **tempo de vida curto**, que deve ser usado quando a spanning tree está em reconfiguração (cujo valor recomendado é de 15 segundos) e que exige um processo de notificação de alterações da topologia da rede (o segundo tipo de mensagem BPDUs).

tempo de vida longo

tempo de vida curto

5. Protocolos de Acesso ao Meio

Duas unidades atrás discutimos uma partilha de um meio de ligação partilhado entre várias máquinas. Lá, vimos uma topologia muito especial - mas será que há mais?

Topologias de rede

Em termos de redes existem quatro **topologias** mais básicas: a topologia em bus, em estrela, em anel e em teia. Sendo uma topologia um modo de disposição dos equipamentos em rede, conseguimos perceber que estas podem ser físicas ou simplesmente lógicas (podendo até existir diferentes topologias a diferentes níveis).

Uma primeira topologia que podemos estudar é a em **teia** (em inglês designada de *mesh*) (Figura 5.1).

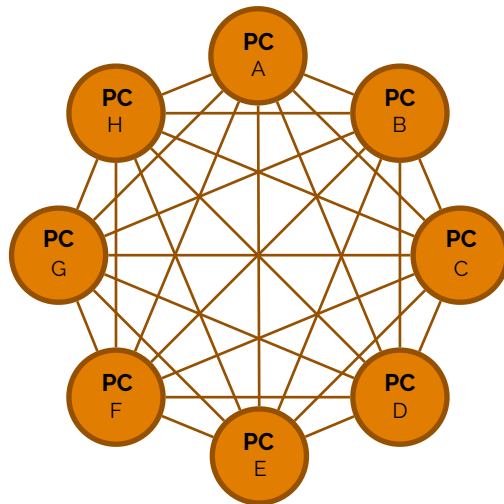
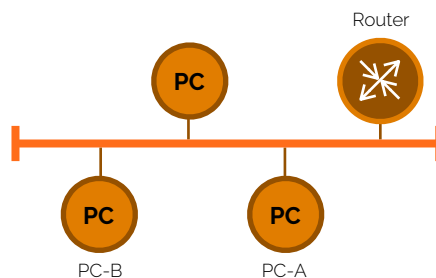


figura 5.1
teia

Quais são as vantagens e desvantagens de uma topologia deste tipo? Por vantagens, podemos referir que há uma excelente disponibilidade de recursos, mas, por outro lado, como desvantagens, temos uma rede muito complexa e difícil de gerir e que cuja complexidade aumenta com o número de ligações a crescer proporcionalmente a n^2 , com n nós na rede.

Uma segunda topologia é a em **bus**. Este modelo está representado na Figura 5.2.



bus
figura 5.2
bus

Neste modelo temos que é fácil ligar um computador ou um periférico e, em comparação à teia, requer muito menos cabos, mantendo-se como uma topologia simples e barata. Mas, da mesma forma que a anterior, esta topologia tem as suas desvantagens e assim, toda a rede fica inativa se existe uma rutura no cabo principal e são necessários terminadores nos extremos do cabo principal. Mais, caso haja algum problema na rede, é difícil identificar a sua origem e a rede inteira deixa de funcionar.

A topologia em **estrela** é outro tipo de topologia mais usado e mais básico. O uso deste modelo beneficia por ser fácil de instalar, não existirem quebras na rede quando se ligam ou desligam equipamentos e por ser fácil detetar falhas e remover equipamentos com anomalias. No entanto, como este modelo se baseia num equipamento central (como um hub), caso este se encontre indisponível, então toda a rede tornar-se-á indisponível também. Em comparação com a topologia em bus, também temos muito mais cabo a ser usado, pelo que se torna mais caro. Uma representação desta topologia é visível na Figura 5.3.

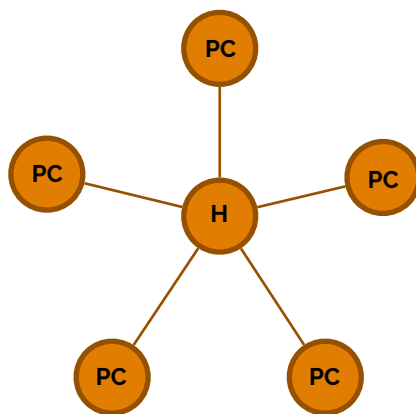


figura 5.3
estrela

Uma alternativa a todas estas topologias é a em **anel**. Este modelo possui um método de acesso ao meio próprio, com garantias de tempo de resposta, sendo simples de controlar e com mecanismos de proteção para falhas no anel. No entanto possui uma complexidade algo elevada e incômoda. Uma representação possível está na Figura 5.4.

anel

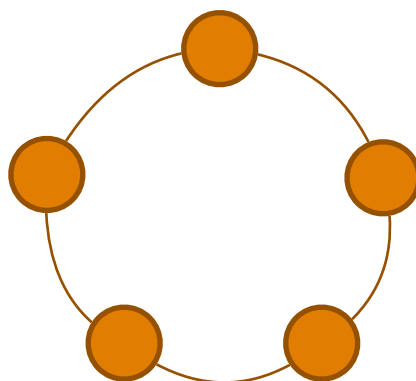


figura 5.4
anel

Apesar das suas desvantagens, a topologia em anel é usada numa tecnologia que gerou um protocolo IEEE, que vamos estudar de seguida.

Token Ring (IEEE 802.5)

O **Token Ring** (protocolo IEEE 802.5) é uma tecnologia de redes que tem como topologia-base a em anel. Aqui há um acesso ao meio em anel com um testemunho, também designado de **token**. Não havendo dados para transmitir no anel, as estações fazem circular o token. Quando uma estação fica com uma trama pronta para transmitir, esta espera pela receção do token, retirando-o do anel e enviando o pacote que pretende. Como o token deixa de circular, as outras estações não podem transmitir (funciona como um sinal de que o meio está ocupado) - deste modo não existirão colisões. O pacote dará então uma volta inteira ao anel e será retirado pela estação emissora. Ao chegar à estação original, a informação passa para o destino e o token é levado de novo, vazio, para o anel.

Vejamos então, como se processa, com mais detalhe. Consideremos a Figura 5.5.

Token Ring
< [IEEE 802.5](#)
token

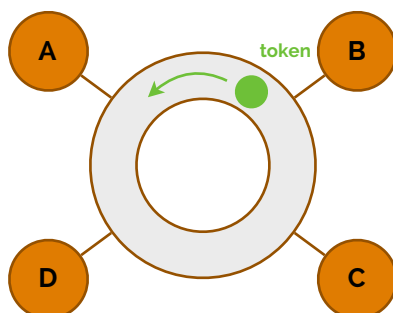


figura 5.5

No Token Ring, ao contrário da Ethernet, o acesso ao meio é puramente determinístico. Uma estação que deseje transmitir um pacote deve então esperar até receber o token que circula no anel. Consideremos então que o equipamento A pretende enviar um pacote c para a rede, de forma a que o equipamento C consiga ter-lhe acesso. Para tal, primeiro A deve recolher o token e deixar c na rede - Figura 5.6.

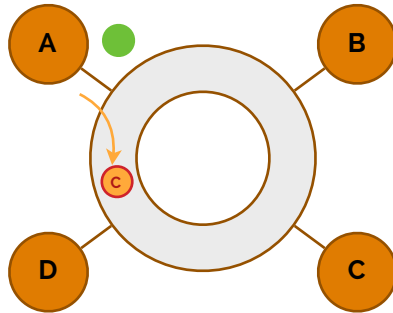


figura 5.6

Entretanto, c circula pelo anel e passa pelo equipamento C, que o copia e, quando chegue à estação emissora é recolhido e esta deixa o token na rede, novamente (Figura 5.7).

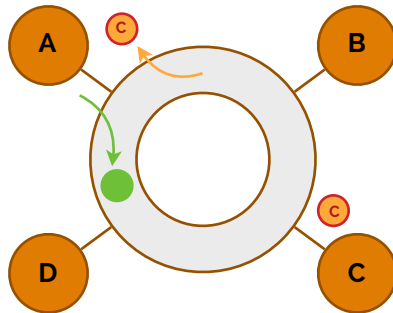


figura 5.7

Nas redes Token Ring deve existir então uma estação responsável pela monitorização do anel - **active monitor**. Caso um anel esteja perdido, então cabe a este active monitor manter um temporizador que deve ser refrescado cada vez que é recebida uma trama ou token, sendo que é inserido um token novo no anel caso o temporizador expire. Este monitor também deve ser objeto de manutenção, pelo que deve enviar periodicamente uma trama *active-monitor-present*, pelo que cada máquina deve, por sua vez, possuir um temporizador, refrescado cada vez que é recebida uma trama deste tipo. Quando os temporizadores expiram, então entra-se num processo de eleição de novo monitor - o novo monitor será a estação ativa com maior endereço.

active monitor

Então mas se a máquina de destino de um determinado pacote copiou-o, como é que a máquina que o enviou sabe que o processo de envio foi feito com sucesso e que o destinatário recebeu bem a mensagem? A mensagem, quando é deixada a circular no anel, mal é copiada é, em parte, modificada, contendo agora uma *flag* para mostrar que a mensagem se encontra num estado "já lida".

Uma grande desvantagem da utilização do Token Ring é que há a necessidade de haverem procedimentos de manutenção do token: tem de ser designada uma estação para garantir que só existe um token no anel e para reinserir um token no anel, caso seja necessário. Também, com tráfego reduzido, as estações têm de esperar que o token passe por elas, até poderem transmitir, contrariamente à Ethernet, em que com tráfego reduzido, a transmissão é imediata. Por outro lado, esta tecnologia também traz as suas vantagens: entre elas, o facto de, com tráfego elevado, o anel funcionar segundo uma disciplina do tipo **round-robin**, garantindo eficiência e equidade é uma delas.

round-robin

Em alternativa a esta tecnologia existem muitas outras - particularmente uma, a que mais se deve usar nos dias que correm - Wireless.

Wireless (IEEE 802.11)

Até agora temos vindo a estudar tecnologias que se baseiam na ligação de equipamento através de fios (ligações físicas materiais). Mas como já sabemos e provavelmente todos usamos, a tecnologia **Wireless** (protocolo IEEE 802.11 - em português sem-fios) não necessita de fios para a transmissão de dados. Contudo, nem tudo nesta tecnologia é tão linear como noutras como a Token Ring. Consideremos o seguinte problema: um computador A e um computador B sentem um canal de transmissão inocupado e transmitem um pacote em simultâneo - será que alguma coisa irá acontecer? Se estivessemos no protocolo Ethernet, o transmissor usaria procedimentos de CSMA/CD para detetar colisões. Agora, no Wireless será que isso acontece? Não. As antenas de Wireless não permitem a deteção de colisões, pelo que precisarão de trabalhar em **half-duplex** (a transmissão e a receção de informação é feita de modo alternado - enquanto que um transmite, o outro só poderá receber informação).

Mas ainda há mais "desvantagens": por exemplo, quanto mais longe estivermos da fonte transmissora de rede, menor será a **força do sinal** (a força do sinal reduz proporcionalmente ao quadrado da distância). Mais, o emissor pode enviar um pacote controlando-o com deteção de colisão e *carrier sense*, mas as colisões podem, contudo, acontecer no recetor. Chegamos assim à conclusão que a deteção de colisões não funciona, no entanto, faz algum sentido usar *carrier sense* - à exceção de um caso, os **hidden nodes**.

Um caso particular que pode acontecer nas transmissões Wireless é o facto de, por vezes, existirem hidden nodes (nós escondidos, em português). Consideremos a Figura 5.8, onde temos um computador A numa área e um outro computador C noutra área, ambos a transmitirem informação para o computador B, que se encontra no meio das áreas, fazendo com que o computador A não saiba que C existe e vice-versa. Isto provoca uma colisão em B, se A e C enviarem informação em simultâneo e nenhum dos dois vai perceber que ocorreu colisão. Uma solução é detetar colisões no recetor, usando um procedimento de **virtual carrier sensing**, isto é, o emissor questiona o recetor se este está a receber pacotes. Caso o recetor não responda, então o emissor deve assumir que o canal está ocupado.

Wireless

< IEEE 802.11

half-duplex

força do sinal

hidden nodes

virtual carrier sensing

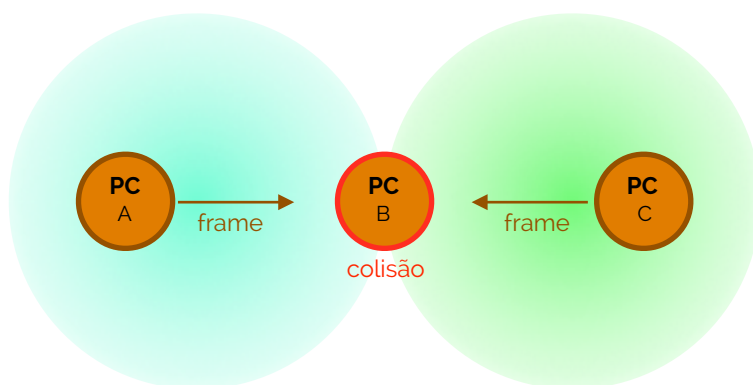


figura 5.8

Outro dilema são os terminais expostos. Para tal, vejamos a Figura 5.9.

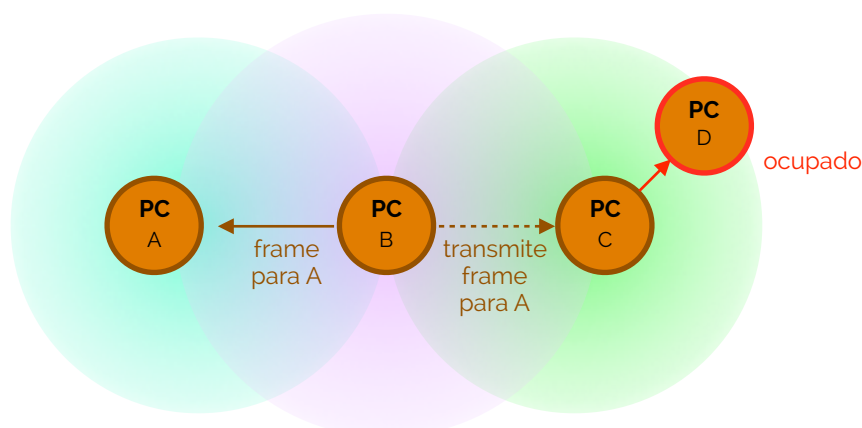


figura 5.9

Na Figura 5.9 temos então uma situação em que o computador B pretende enviar um pacote para A. Para fazer isto num rede Wireless o computador envia o frame para a rede, não de forma orientada simplesmente para o computador A. Assim, há o risco do pacote cair num terminal que lhe seja acessível, mas cujo pacote não lhe interessa - risco de encontrar **terminais expostos**.

Uma forma de resolver estes problemas, de forma genérica, é através do **MACA** (acrónimo de *Multiple Access with Collision Avoidance*). Usando o MACA todos os terminais terão de respeitar o prévio envio de pedidos e permissões para ocupar os canais. Assim, quando um terminal pretende enviar um pacote para outro deve, primeiro, enviar um pequeno pacote denominado **RTS** (em inglês *Request to Send*) como um pedido para o envio de um pacote. Em resposta a este pedido, o terminal de destino deve também ser capaz de enviar outro pacote, denominado de **CTS** (em inglês *Clear to Send*), de forma a indicar que o canal não está ocupado e que o primeiro terminal pode então enviar os seus pacotes.

Estes pacotes de RTS e CTS, ambos contêm um endereço de destino, de origem e um tamanho especificado da informação a ser enviada. Vejamos então como é que a prevenção de colisões (não é deteção!) se comporta em termos de hidden nodes (Figura 5.10).

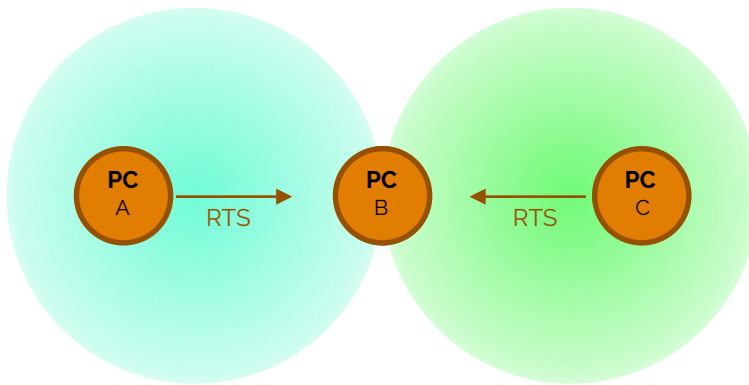


figura 5.10

Então primeiro ambos os computadores A e C devem enviar um pedido RTS de forma a saber se o computador está disponível para receber informação nova. Consideremos que o computador B aceita a comunicação de A. Para tal, este envia um CTS para A (Figura 5.11) e aguarda pelo pacote.

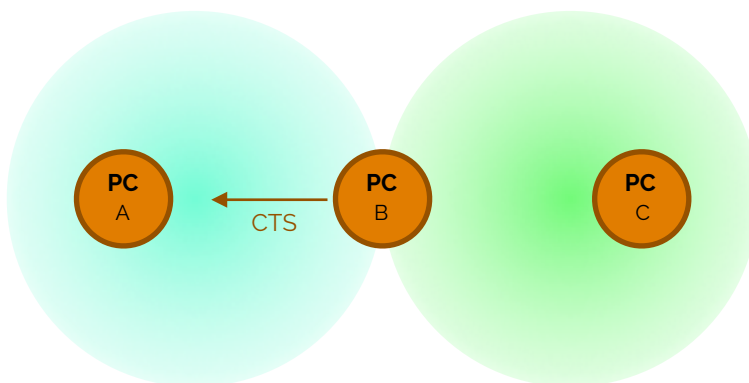


figura 5.11

No final da transmissão do pacote que A pretendia enviar para B, B envia um pequeno pacote **acknowledge** para informar A que recebeu toda a informação pretendida. De seguida irá responder ao computador C, que entretanto ficou à espera da disponibilidade de B.

Em relação aos nós expostos, também podemos verificar que com o MACA já se tornam possíveis as comunicações sem efeitos colaterais. Consideremos então a Figura 5.12.

acknowledge

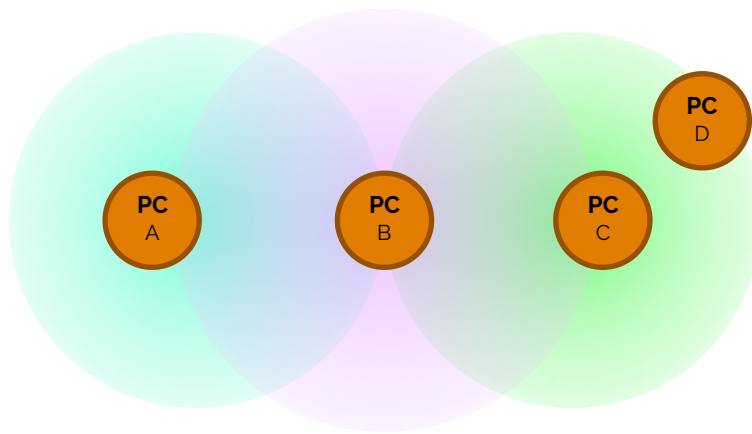


figura 5.12

Consideremos então que o computador B pretende enviar um pacote para A - o que é que acontece? Se isto acontecer, primeiro B deverá enviar um pedido RTS para A, mas como o envio não é orientado, ele enviará em todas as direções, pelo que C também irá receber um pedido CTS para A. Como C não é A, então este pedido propagar-se-á para os seus vizinhos, fazendo com que D receba um RTS de C (Figura 5.13).

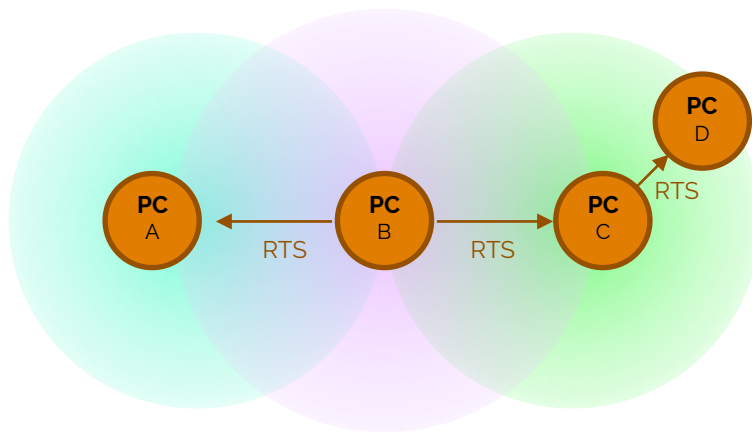


figura 5.13

Como os pedidos RTS são pacotes com informações acerca de quem envia e quem recebe, só A é que vai enviar um CTS para B, como demonstra a Figura 5.14. De seguida o pacote pretendido é transmitido e, quando terminar, A envia um acknowledge para B, informando-o que já terminou de receber.

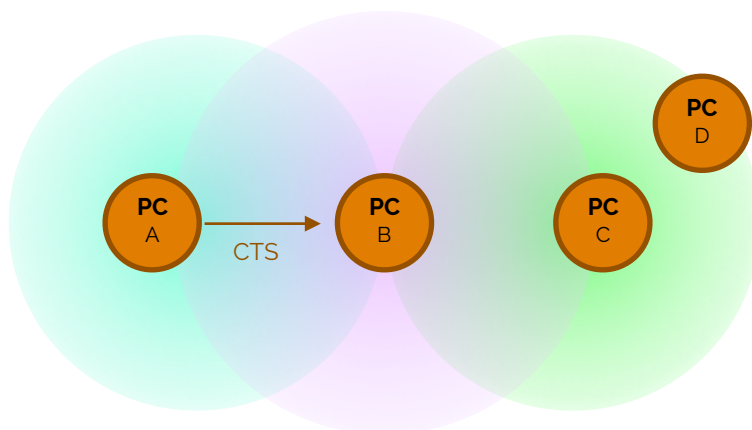


figura 5.14

6. Camada de Rede e endereçamento IP versão 6

Até agora já estudámos conceitos da camada 2 da estrutura de uma rede. Agora iremos dar entrada no estudo da camada de rede mais lógica - a **network layer**.

network layer

Endereçamento privado

Alguns dos endereços IPv4 que estudámos, de facto, não podem ser usados de um modo aleatório, isto é, os acessos à utilização de alguns endereços estão condicionados. Isto acontece porque, para garantir, de certa forma, uma extensibilidade maior por parte dos endereçamentos IPv4, criou-se o conceito de endereçamento público e endereçamento privado. O **endereçamento privado** é então um conjunto de gamas de endereços IPv4 destinados para redes locais, privadas. O facto de serem privadas, faz com que todos os pacotes cujos endereços de destino sejam da gama dos privados não possam circular na rede pública. Na Figura 6.1 podemos ver quais são as gamas a que correspondem os endereços privados.

endereçamento privado

prefixo	endereço mais baixo	endereço mais alto
10/8	10.0.0.0	10.255.255.255
172.16/12	172.16.0.0	172.31.255.255
192.168/16	192.168.0.0	192.168.255.255
169.254/16	169.254.0.0	169.254.255.255

figura 6.1
endereçamento privado

Estes endereços, por exemplo, são os que temos nas nossas casas. Contudo, há um problema. Como é que em nossas casas, nós conseguimos, com um endereço privado, aceder à rede pública, melhor, à maior rede pública que existe - a Internet?

Network Address Translation (NAT)

Quando estamos nas nossas casas, dentro da nossa rede temos um endereço próprio e, se nos instalarmos noutra casa e acedermos à rede dessa, até podemos ficar com o mesmo endereço. Isto acontece porque os endereços que nos são atribuídos pelos nossos equipamentos na nossa rede de casa são privados e o contexto de existência destes endereços são únicos de casa em casa. Mas então como é que conseguimos aceder a redes públicas? Antes dissemos que todos os pacotes cujos endereços de destino são privados não podem circular na rede pública. O que acontece é que tem de existir um mecanismo que converta endereços privados em endereços públicos, e vice-versa, à entrada/saída da nossa rede. A esse mecanismo damos o nome de **NAT** (acrónimo inglês para *Network Address Translation*).

NAT

Consideremos a Figura 6.2. Todos os datagramas que surtam do interior da rede 10.0.0.0/24 terão, como já nos deve ser claro, um destino ou uma origem 10.0.0.0/24. Mas o que é que acontece se quisermos enviar um pacote para fora? Neste caso, quando o datagrama chega ao router, este deve ser encarregue de modificar a trama de forma a que a origem seja agora um endereço público, como 138.76.29.7.

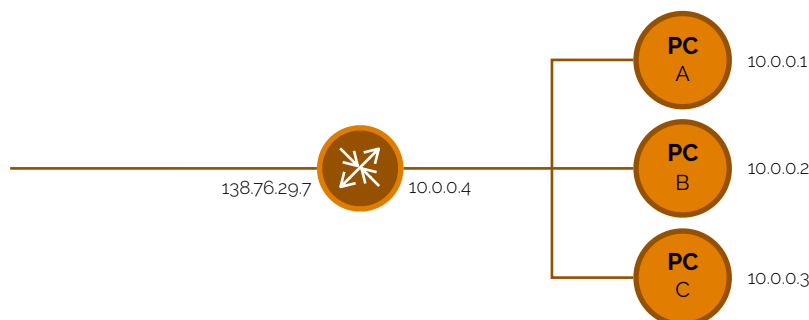


figura 6.2
NAT

Então mas se nós, de fora, pretendemos enviar um pacote para o 10.0.0.2 como é que fazemos? Não podendo enviar para 10.0.0.2, mas antes para 138.76.29.7, como é que depois se distingue o computador B? Para isso existem os **portos**.

portos

Consideremos então um caso em que fazemos um ping da nossa rede 10.0.0.0/24 para fora (para o 128.119.40.186), desde o computador A. O computador A tem o endereço 10.0.0.1 e cria, então, um datagrama com destino 128.119.40.186. A estes destinos e origens têm de estar associados portos: ao computador A está associado o porto 3345 e ao endereço externo da rede está associado o porto 80. Quando este datagrama chega ao router, o router deve ter, numa **tabela NAT**, a correspondência entre o endereço privado e o endereço público. Consideremos que a tabela da Figura 6.3 é a nossa tabela NAT.

tabela NAT

WAN	LAN
138.76.29.7. 5001	10.0.0.1. 3345

figura 6.3

Dado que existe correspondência entre o endereço privado e um endereço público com um determinado porto (neste caso é 5001), o router vai modificar o cabeçalho do datagrama e agora o processo funciona como já estudámos anteriormente. Quando o pacote ICMP chega ao destino, este envia uma resposta e, quando esta chega ao router vem com as informações de um destino 138.76.29.7, 5001. O router, ao ver esta informação verifica na tabela NAT a que corresponde o porto 5001 e muda o cabeçalho do datagrama para 10.0.0.1, 3345.

Se repararmos bem, esta pequena modificação que fizemos no tratamento de endereços IPv4 fez com que não fosse assim tão fácil terminar com todas as combinações possíveis para endereços atribuídos em 32 bits. Fazendo com que cada área local tenha um e um só endereço IP para se identificar na rede pública, juntamente com o desígnio de várias outras entidades através de portos permite, também, maior mobilidade e modularidade. Na verdade, mais que 60000 ligações em simultâneo podem ser feitas a partir de um só endereço público IP.

Dynamic Host Configuration Protocol (DHCP)

Consideremos a seguinte situação: nós temos um computador cujo nome é R e pretendemos ligar a uma determinada rede de área local. Para fazermos isso, já devemos saber que não chega ligar o cabo ao nosso computador. Temos de configurar um IP, por exemplo. Mas será que existe uma forma de contornar este problema? Sim, para isto temos um processo denominado de **DHCP**. O DHCP (sigla inglesa para *Dynamic Host Configuration Protocol*) tem como objetivo permitir que os terminais que se associem a uma rede recebam, dinamicamente, um endereço IP por um servidor. Este empréstimo de endereço, enquanto a ligação está feita, deve ser constantemente renovada, de tempo a tempo.

DHCP

Vejamos como funciona o DHCP: o computador R liga-se à rede; de forma a receber um endereço IP este deve lançar para o servidor DHCP uma mensagem (trama) **DHCP discover** (esta mensagem terá como origem o endereço 0.0.0.0, 68 - dado que ainda não tem endereço -, destino 255.255.255.255, 67 - dado que ainda não sabe onde é que está o servidor DHCP, dirigindo a mensagem para o porto 67 de servidor DHCP -, e endereço próprio 0.0.0.0 - campo *yiaddr* de "your address" vazio porque ainda não tem endereço; de seguida cabe ao servidor DHCP responder ao cliente, fazendo-lhe uma oferta com um **DHCP offer**, com um endereço de origem 223.1.2.5, 67 - endereço do servidor, por exemplo -, com um endereço de destino 255.255.255.255, 68 - dado que o cliente DHCP, representado pelo porto 68, ainda não tem endereço -, propondo o endereço 223.1.2.4 - no campo *yiaddr* - com um tempo de vida para resposta de 3600 segundos, por exemplo; agora cabe ao cliente dizer se pretende ou não pedir esse endereço, pelo que para tal segue uma nova mensagem, desta vez um **DHCP request** pedindo o endereço, em *yiaddr*, 223.1.2.4, ao que o servidor responde, positivamente, num **DHCP acknowledgement**. Esta transação está representada de forma esquemática na Figura 6.4.

DHCP discover

DHCP offer

DHCP request

DHCP acknowledgement

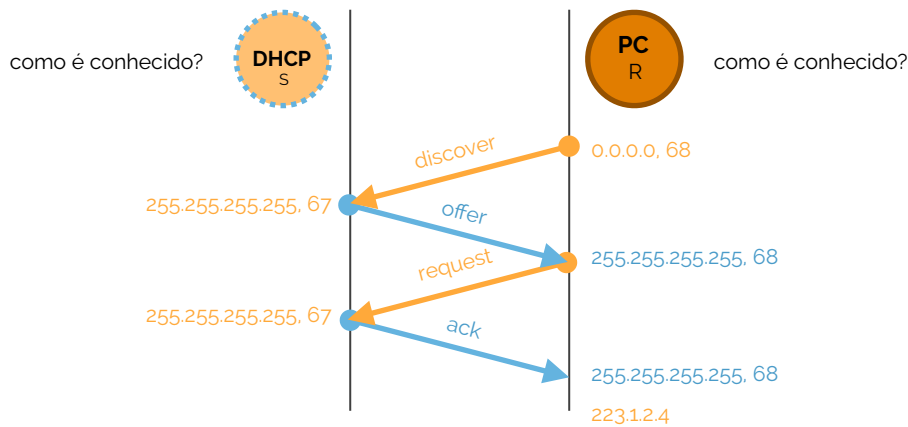


figura 6.4
DHCP

Então mas de onde é que vêm os endereços e o que é que acontece quando se deixa de usar um dos endereços emprestados? São duas boas questões e que o DHCP está preparado para responder. Quando à primeira, isso reverte para a definição de um **servidor DHCP** - equipamento que se identifica num porto (67) e que distribui endereços num conjunto de endereços (uma **pool de endereços**). Quanto à segunda questão, quando um endereço deixa de ser usado, então um determinado tempo (**lease time**) que cada endereço possui, quando se esgotar e não for renovado, passa a estar disponível para quem precisa.

servidor DHCP
pool de endereços
lease time

O cliente DHCP não tem de ter informações prévias acerca da rede quando arranca. Por isso, no primeiro pacote que envia para a rede, ele não conhece nem o seu próprio endereço IP, nem o endereço da rede onde está, tal como o endereço do servidor DHCP. A primeira mensagem que é enviada (DHCP discover) é uma mensagem do tipo **BootP**. Como o próprio nome o indica, esta mensagem serve para descobrir os servidores de DHCP que existem na rede. O cliente pode também indicar neste pacote qual o endereço IP que pretende alugar. Os servidores DHCP que recebem o pedido respondem com uma mensagem do tipo DHCP offer. Essa mensagem também é encapsulada num pacote de resposta do tipo BootP. O pacote BootP tem que ser enviado para o endereço de broadcast, uma vez que o cliente ainda não possui endereço IP. Nessa resposta cada servidor diz ao cliente que endereço IP lhe pode fornecer. Os servidores tentam sempre respeitar a preferência do cliente por um determinado endereço. O cliente recebe as respostas dos servidores. Se algum dos servidores lhe oferecer o endereço IP que pediu, então escolhe-o. O cliente envia então um pacote do tipo DHCP request ao servidor que lhe cedeu o endereço. Quando o servidor recebe o pedido, reserva o endereço IP para o cliente e responde com um DHCP acknowledge.

BootP

O **leasing** dos endereços é feito segundo três parâmetros: **renewal time** que, sendo metade do tempo de leasing (empréstimo), define-se por ser a altura em que o cliente deve tentar renovar o seu empréstimo no servidor que lhe cedeu o endereço; **rebinding time** que, sendo 85% do tempo de leasing, o cliente deve tentar renovar o seu empréstimo do seu endereço em qualquer servidor DHCP disponível, caso não tenha conseguido antes; e **leasing time** que, claro está, é o tempo total de empréstimo que, caso não seja renovado, faz com que o cliente perca o endereço.

leasing, renewal time

rebinding time

leasing time

Existem outras mensagens de DHCP, entre as quais:

- › **DHCP decline:** o cliente rejeita a oferta de um servidor e reinicia a aquisição de endereço IP; **DHCP decline**
- › **DHCP nack:** o servidor informa que não pode renovar o empréstimo de um endereço; **DHCP nack**
- › **DHCP release:** o cliente informa que o servidor que já não está interessado no endereço; **DHCP release**
- › **DHCP inform:** o cliente pede informação adicional (neste caso, o cliente tem um endereço IP mas requer, por exemplo, um endereço de um servidor DNS). **DHCP inform**

Endereçamento IP versão 6 (IPv6)

Com o endereçamento IPv4 tínhamos um grande problema - dado que o seu tamanho era de 32 bits, isto é, 4 bytes, apenas poderíamos, sem repetições ou outros processos, endereçar aproximadamente 4 294 967 296 máquinas. Uma forma de resolver este problema é simplesmente criando um novo esquema de endereçamento.

Mas este foi apenas um dos inconvenientes que vieram a fazer com que se criasse um novo esquema de endereçamento, que tornasse as comunicações mais simples. Surgiu assim o **IPv6**. Enquanto que o IPv4 tem um tamanho de 32 bits, o IPv6 vem quadruplicar este tamanho, logo, tem 128 bits. Mais, com este endereçamento é possível haver maior eficiência em termos de roteamento, desempenho e escalabilidade de taxas de encaminhamento.

A **representação**, dada a extensão do endereço, é feita em números hexadecimais de 16 bits separados por "dois pontos" (:). Por questões de simplicidade, é possível abreviar, quando se apresentam blocos contíguos de zeros, escrever "::". Por exemplo, enquanto que podemos escrever o endereço 2001:0DB8:0000:130F:0000:0000:087C:140B, podemos apenas escrever 2001:DB8:0:130F::87C:140B.

O **cabeçalho IPv6** é diferente do cabeçalho IPv4, dado que este possui 40 bytes sem checksum e opções e com campos novos (**flow** sem uso corrente, mas com utilidades dependendo do caso e outros cabeçalhos de opção). Em comparação com o cabeçalho de um datagrama IPv4 (como o representado na Figura 1.7), eis o cabeçalho IPv6 (Figura 6.5).

IPv6

representação

cabeçalho IPv6

figura 6.5

cabeçalho IPv6



Em alternativa ao campo de opção do endereço IPv4, no IPv6 temos **cabeçalhos de extensão**, que são adicionados conforme é necessário. Alguns destes cabeçalhos são: os *hop-by-hop, routing, fragment, destination options, authentication, encapsulating security, ...*

cabeçalhos de extensão

Os endereços IPv6, para além da sua complexidade, fornecem, pelo contrário, bastante simplicidade aos equipamentos, para trabalharem em rede - uma das vantagens do seu uso é o facto de serem auto-configuráveis, e de possuírem um endereço local.

Um endereço desta versão é composto por 64 bits de endereço de rede + 64 bits de endereço da **interface** de ligação. Em termos de endereço de rede, convém frisar que há especificidades próprias para a ordenação - os endereços têm alcance. Por **alcance** definimos quatro possíveis: global, link-local, unique-local e multicast. Um alcance **link-local** é apenas válido dentro da mesma LAN ou ligação e o endereço começa sempre por

alcance

link-local

FE80::/10. Um alcance **unique-local** é válido dentro de um domínio privado e não pode ser usado na Internet, tendo um endereço FC00::/8 ou FD00::/8. Quando vemos um endereço começado em 2₁₆, então significa que estamos perante um alcance **global**. Por fim, endereços começados por FF00::/16 são **multicast**. Outra característica que os endereços IPv6 têm é um **tempo de vida**.

Então mas como é que funcionam os 64 bits de interface? Como é que se constroem? Dado que se tenta identificar a interface de ligação, então não existe nada melhor do que o endereço MAC da interface a que se vai ligar para ter como base dos nossos 64 bits. Para tal, apenas se tem de inserir, a meio do endereço MAC, os bytes indicadores FF-FE e modificar o sétimo bit do MAC para '1' se for uma interface única ou '0' se não o for. Isto é o modo de configuração da interface proposto pelo protocolo **EUI-64**. Outras formas de configurar passam por gerar um número pseudo-aleatório, designar um valor por DHCP ou configurar manualmente.

Uma das características mais importantes, como já referimos, é a **auto-configuração**. Quando um terminal se liga à rede este deverá ser capaz de obter a sua configuração correta de forma a associar-se. Se isto acontecer, automaticamente esta alteração na rede será propagada por via de mensagem para todos os outros equipamentos ligados. Existem assim dois métodos de configuração automática: o método **stateless**, onde a configuração é determinada pela rede; e o método **stateful**, onde a configuração é determinada pela gestão da rede.

Todos estes mecanismos processam-se da seguinte forma:

- é criado um link-local;
- verifica-se a unicidade do endereço de link-local (através de algoritmos de DAD - *Duplicate Address Detection*);
- seleciona-se o método de configuração;
- determina-se a informação a ser auto-configurada (endereços, gateways, ...).

Se no processamento for escolhido o método de configuração **stateless**, então o terminal irá escolher o seu próprio endereço, combinando informação local - como o endereço MAC - e informação que é dada pelos routers. Este processo traz vantagens, dado que não há necessidade de configurar manualmente os terminais, tendo apenas de abordar minimamente os routers e não havendo a necessidade do suporte de servidores. Mas o que é que acontece se não houver router? Caso isto aconteça, então o terminal tem de criar o seu próprio endereço link-local, que é suficiente para que se permita efetuar uma comunicação no mesmo segmento de rede local.

Por outro lado, se no processamento for escolhido o método de configuração **stateful** esta será feita baseada num modelo de cliente-servidor, através do **DHCPv6**. Este modelo permite então a alocação de endereços IPv6 para os terminais e a entrega de informações de configuração específicas de cada terminal. Isto garante que haja um maior controlo de configuração e suporta conceitos de configuração automática do IPv6 (como a automática mudança de endereços).

7. Detecção e Controlo de Erros

Em redes, o grande interesse (grande feito) é conseguir estabelecer vias de transmissão entre equipamentos. Para tal criamos canais de comunicação o que por si, não impede que hajam **erros**. Quando falamos em erros podemos referir várias consequências provenientes destes, como pacotes corrompidos, perdidos ou recebidos fora de ordem. Mas será que existe alguma forma de tentar melhorar os nossos sistemas de forma a que possam evitar, corrigir e detetar erros? Até agora já estudámos mecanismos de prevenção ao erro - mais precisamente às colisões (CSMA/CD).

Uma forma de criar condições de controlo de erros é através de mecanismos de **handshaking** entre um emissor e um recetor - protocolos de retransmissão **ARQ** (acrónimo

unique-local

global

multicast

tempo de vida

EUI-64

auto-configuração

stateless

stateful, DHCPv6

handshaking, ARQ

inglês para *Automatic Repeat and reQuest*). Estes protocolos podem então ser necessários em diferentes camadas protocolares da organização da rede. A partir deste ponto vamos admitir os seguintes conceitos:

- **pacote**: conjunto de bits entregues para transmissão pela camada de rede;
- **trama**: conjunto de bits entregues para transmissão à camada física;
- quando são detetados erros numa trama, é transmitida uma nova trama contendo o pacote anterior.

pacote
trama

Juntamente com os conceitos acima detalhados assumimos que:

- os erros das tramas são sempre detetados;
- as tramas podem sofrer atrasos variáveis mas limitados;
- algumas tramas podem ser perdidas;
- as tramas chegam na ordem em que foram transmitidas.

Stop and Wait (SW)

Um algoritmo de deteção e controlo de erros é o **Stop and Wait** (SW). Este algoritmo processa-se da seguinte forma: um computador A pretende enviar três tramas (a 1 e a 2) para o computador B. Para tal, primeiramente envia a trama 1, esperando que esta chegue a B e este envie uma mensagem de reconhecimento. Só depois de ter recebido a mensagem de reconhecimento é que se está em perfeitas condições de enviar a segunda trama (trama 2). Este procedimento está representado na Figura 7.1.

Stop and Wait

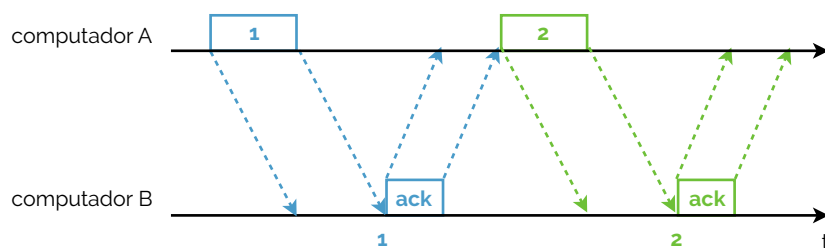


figura 7.1

Então e se ocorrer um erro? O que significa ocorrer um erro? Ocorrer um erro significa que a trama 1 foi enviada pelo computador A e este nunca mais recebeu o reconhecimento ou receber um reconhecimento de algo antigo ou irreconhecível. Consideremos então que enviamos uma trama do computador A para o computador B. Então a trama é enviada para B e B deve iniciar um reconhecimento - mas vamos imaginar que desta vez, não houve reconhecimento. O que é que acontece? Com o envio de tramas deve estar associado um tempo até o reconhecimento. Se este tempo for ultrapassado sem que o reconhecimento chegue ao computador A, então repete-se o envio. Esta situação está representada na Figura 7.2.

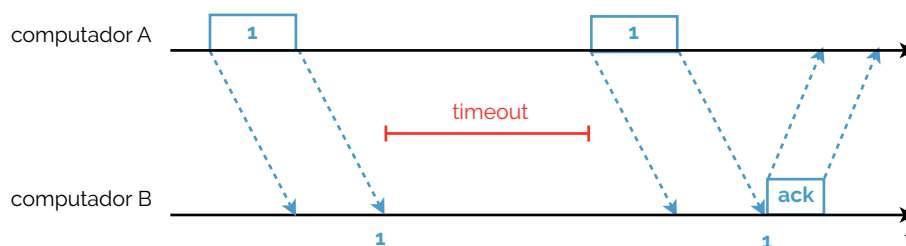


figura 7.2

A segunda alternativa de erro também pode acontecer, isto é, se o computador A envia uma trama 1 para o computador B, mas só fica a saber depois do tempo de timeout ter sido contado e do computador A ter reenviado a trama. Isto é fácil resolver - basta incluir uma numeração de sequência nos reconhecimentos (*ack*). Assim, podemos melhorar

este nosso algoritmo se, na trama a ser enviada designarmos o seu número como um número de sequência (**SN**) e, no reconhecimento designarmos um segundo número que será denominado de **request number** (RN), o qual fará respeito à próxima trama a receber. Na Figura 7.3 está um exemplo de aplicação desta solução.

SN
request number

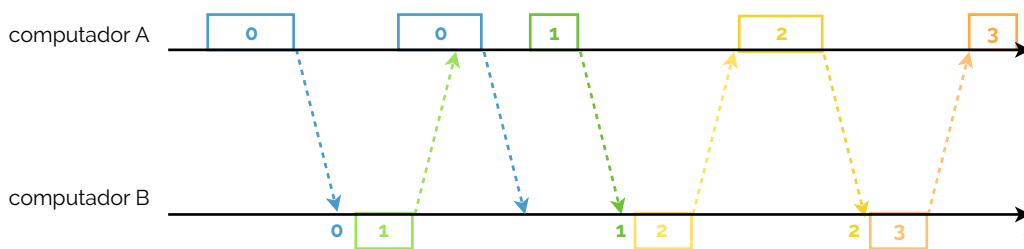


figura 7.3

Agora, em termos de simplicidade de conteúdo, será que faz algum sentido gastar mais bits para armazenar o número de sequência? Quantos estados diferentes temos de trama em trama? Só dois, pelo que podemos codificar com '0' e com '1'.

Já em termos de desempenho podemos analisar como é que se comporta o algoritmo de stop and wait. Se verificarmos, para além do facto do nosso transmissor estar inutilizado enquanto aguarda por um reconhecimento é mau - é uma má gestão de recursos. Mais, se houver um erro na transmissão ter-se-á que repetir uma só trama, até que esta chegue intacta ao computador B e este envie um reconhecimento para A.

Go-Back-N (GB-N)

Uma alternativa melhor ao algoritmo de Stop and Wait é o **Go-Back-N** (GB-N). Aqui já é permitido ao emissor enviar mais do que uma transmissão antes de receber o reconhecimento da primeira. Cria-se assim o conceito de **janela de n tramas**, que consiste no número máximo de tramas que o emissor pode transmitir antes de receber o reconhecimento da primeira. Na mesma, após um timeout, o emissor que não tenha recebido o reconhecimento após uma trama n , re-transmite essa trama e também todas as seguintes. Assim, após receber a trama n , o recetor apenas aceita receber a trama $n+1$ e descarta a trama se tiver um número de sequência superior, isto porque, se uma trama não chegou ao destino, então essa trama e as seguintes serão re-transmitidas e porque assim também se garante uma maior simplicidade do recetor. Já no envio de um reconhecimento de uma trama n , está implícito que todas as tramas até n já foram recebidas. Vejamos então a Figura 7.4, onde se tenta representar um exemplo de transmissão com GB-7 (onde o n é igual a 7).

Go-Back-N

janela de n tramas

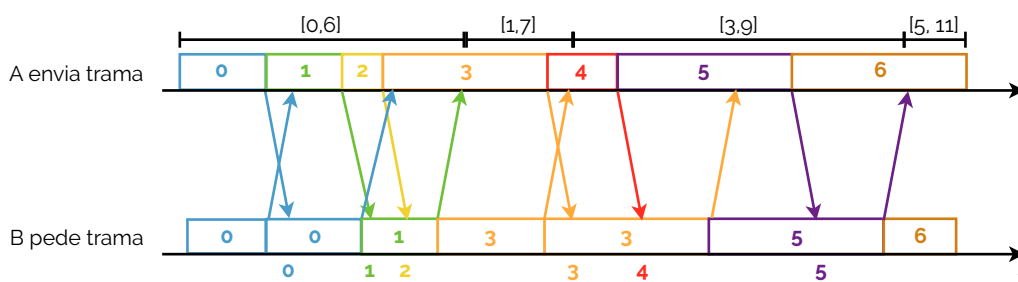


figura 7.4

Como podemos reparar na Figura 7.4, o computador A envia a trama 0 e, ao mesmo tempo, o computador B pede a mesma trama. Como ambos os computadores se encontram a uma determinada distância um do outro, há um atraso no envio (representado pelas setas diagonais), o que faz com que o envio de 0 só tenha sido recebido num novo pedido de 0 por parte do computador B. Mal este 0 seja recebido por B, ele fica guardado e, quando acabar de pedir o 0 que já tinha iniciado antes, irá iniciar o pedido do seguinte - o 1. Da parte de A, como o 0 foi enviado, então não há perigo nenhum de iniciar o envio de uma nova trama, com sequência seguinte - envia então o 1. Este envio é recebido um pouco depois de B ter iniciado o seu pedido. Quando A termina de enviar 1,

começa a enviar, pelas mesmas razões de antes, 2 e este chega B enquanto este termina de pedir 1 - ora, isto não tem qualquer perigo, dado que se trata do número de sequência imediatamente seguinte. Assim sendo, quando B terminar de pedir 1, irá pedir o seguinte àquele que recebeu por último - o que irá incrementar a janela de pedidos de 0 a 6 para 1 a 7 - logo, como o último a ser recebido foi o 2, então irá pedir o 3. Enquanto B pede o 3, A já antes tinha iniciado o seu envio. Dado o seu tamanho de trama, 3 demora a ser enviado e, não tendo chegado no primeiro pedido por parte de B (que provoca uma alteração do valor da janela, dado que B garante já ter recebido tudo até 3), este pede 3 de novo, aguardando pela sua chegada. Eis então que 3 chega a B, acompanhado, instantes depois por 4, que fora enviado por A logo a seguir a 3. Dado que 4 já foi recebido por B, então a próxima trama a pedir será a 5, a qual já teria sido iniciada por A para envio e recebida depois por B. Quando o pedido de 5 chega a A, este poderá alterar os limites da janela para de 5 a 11.

Vejam agora um caso ligeiramente diferente, onde expomos um Go-Back-4 (Figura 7.5).

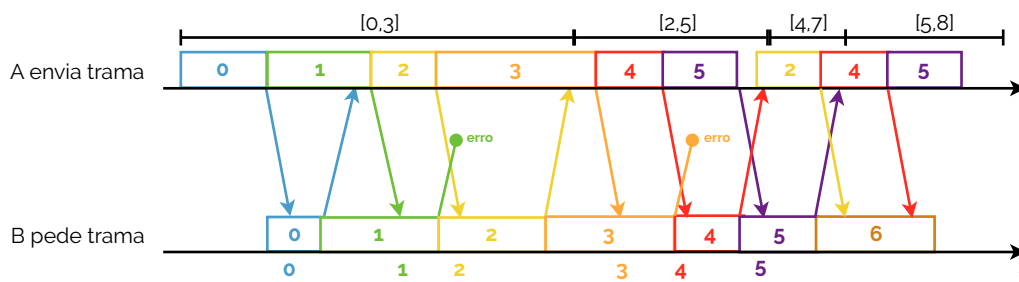


figura 7.5

Na Figura 7.5 o computador A envia uma trama com número de sequência 0, a qual é recebida no tempo do computador B processar o pedido da mesma - é então recebida a trama. No final do envio de 0, A envia diretamente a trama 1, enquanto B termina de pedir 0 e começa a preparar o pedido de 1 - esta trama também é recebida por B. Mas agora algo estranho acontece e o pedido que B formulou de 1, não chegou a A. Do lado de A, não sabendo se B recebeu ou não 1, prossegue o envio de 2 (número de sequência seguinte), a qual é devidamente recebida por B e confirmada a A (o que faz com que a janela se altere para 2 a 5), enquanto que o envio de 3 está a ser processado. Então mas se A não tem confirmação direta que 1 chegou a B, como é que a janela pode mudar para de 2 a 5? Acontece que pelo facto de A receber a confirmação de que B já tem 2, então, por inferência, tem-se a garantia que já tem também todos os números de sequência anteriores (onde também se encontra o 1). Continuando, 3 é enviado de A para B, tendo B recebido 3 e há uma falha no envio da confirmação de 3 em B. No entanto, 4 já estava a ser enviado e chegou a B (é enviada a confirmação, que faz alterar a janela para de 4 a 7, pelas mesmas razões de antes). Mas agora, de novo, algo de estranho acontece - cria-se um compasso de espera entre o envio de 5 e o envio de 2 e, mais, porque é que voltámos ao envio de 2? A janela que abrange o envio de 4 e 5 é de 2 a 5, logo, se não se recebe a confirmação do envio do limite máximo da janela, então volta-se ao último número confirmado - neste caso, o 2 -, após um pequeno timeout. Voltando ao 2, este é enviado (embora já recebido por B) e a janela muda para de 4 a 7, pelo que se envia, sequencialmente, as tramas de 4, 5, ... de A para B.

Selective Repeat (SR)

Com esta alternativa de controlo de erros denominada de **Selective Repeat (SR)** é permitido ao emissor enviar mais que uma trama antes de receber a confirmação da primeira, gerando-se assim uma janela N , em que N é o número máximo de tramas que o emissor pode transmitir antes de receber a confirmação (ACK) da primeira. Após um determinado timeout, o emissor que não tenha recebido a confirmação de uma trama $SN = n$, retransmite apenas essa trama - isto acaba por ser mais vantajoso que no GB-N porque em GB-N envia-se uma sequência de tramas, enquanto que aqui apenas se enviam

Selective Repeat

as que faltam. Após receber todas as tramas até ao $SN = n$, o recetor aceita receber qualquer trama cujo número de sequência (SN) seja de $n + 1$ até $n + N$. Em termos gerais, apenas as tramas perdidas são retransmitidas, pelo que podem ser recebidas fora de ordem, sendo que o buffer de receção permite a reordenação pela ordem original. Da parte do recetor tem então de ser especificado o request number (RN) que deve ser igual a $SN + 1$ e P , sendo este o número de sequência da trama com SN mais elevado corretamente recebida.

Deteção de erros

Por vários motivos, uma comunicação que é feita em meio computacional, dados os acontecimentos de manipulação de sinais, podem ser sujeitos a efeitos nefastos que podem gerar erros nas transmissões. Como vimos em Arquitetura de Computadores II (a2s2), existem várias formas de detetar e corrigir erros, desde os códigos de paridade a códigos de Hamming. Em redes o nível de prejudicialidade é bastante alto, em termos de erros nas comunicações, pelo que é conveniente saber não só encontrar métodos fortes de deteção e correção de erros, como também rápidos e eficazes.

Os problemas que afetam as comunicações, no panorama desta disciplina, são vulgarmente pacotes corrompidos, perdidos ou recebidos fora de ordem. Os métodos que iremos abordar são muito simples, mas bastante usados nos dias que correm.

© Richard Hamming

Código de verificação de paridade

Como já tivemos oportunidade de ver noutras disciplinas de sistemas digitais, é possível (e muito usado) designar um bit como um **bit de paridade**, que conterà informação acerca de uma dada trama e da sua composição. Basicamente, no fim de cada palavra binária (8 bits) adicionamos um bit extra que será '0' se o número de 1's na palavra for par e '1' se for ímpar. Vejamos então um exemplo na Figura 7.6.

código de paridade

o	1	0	0	0	0	0	1	carater ASCII 'A'
o	1	0	0	0	0	1	1	com 1 bit errado
o	1	0	0	0	1	1	1	com 2 bits errados

figura 7.6

Para verificar se existe erro na transmissão calcula-se a paridade da palavra no recetor e compara-se com o bit de paridade que vem do emissor. Assim, no primeiro caso de erro, a paridade da palavra é 'ímpar' (logo '1'), o que não coincide com o valor '0' do bit de paridade (há erro). Já no segundo caso de erro, a paridade é 'par' - tem 4 bits a '1' -, mas o bit de paridade é '0', logo, "não há erro" - isto está mal, mas acontece porque o código de verificação de paridade simples como este apenas deteta todos os erros com número ímpar de bits errados - todos os erros com número par de bits errados não são detetados. Cria-se assim a deteção de erros por paridade em **blocos**. Para tal é formado um bit de paridade por cada palavra binária individual (na horizontal) e também sobre o bloco de palavras (na vertical). No fim do bloco adiciona-se um carater chamado de **block-check character** (BCC). Na Figura 7.7 está um exemplo desta técnica.

blocos

block-check character

A	o	1	0	0	0	0	0	1
B	o	1	0	0	0	0	1	0
C	1	1	0	0	0	0	1	1
BCC	1	1	0	0	0	0	0	0

figura 7.7

Cyclic-Redundancy Check (CRC)

A forma mais usada para deteção e correção de erros nas tramas em redes é através de **códigos CRC**, como já vimos antes na disciplina de Introdução aos Sistemas Digitais (als1).

O algoritmo para gerar um código CRC de n bits usa um gerador polinomial composto por $n+1$ bits que funciona como divisor módulo 2 da mensagem original. O resto da divisão R é o código CRC gerado. Sejam M os dados a transmitir (com número arbitrário de bits), G o polinómio gerador de grau n (com $n+1$ bits) e R o resto da divisão módulo 2 de M por G (com n bits). A mensagem a transmitir calcula-se através da soma $M + R$. G é conhecido por ambos receptor e transmissor. A escolha do G determina quais os tipos de erros que são detetados.

Por exemplo, imaginemos que $M=11100110$, $G=11001$ ($n=4$) e CRC₄: $G=x^4+x^3+x^0$. O algoritmo é o seguinte:

- Adicionar n zeros à parte menos significativa de M : 11100110 0000 (só no transmissor);
- Alinhar o polinómio gerador com o bit mais significativo de M que está a 1;
- Gerar uma nova palavra realizando a operação XOR entre M e G ;
- Se M resultante é diferente de 00...0, ir ao ponto 2;
- Se $M = 00...0$, os n bits menos significativos contêm R .

8. Encaminhamento em redes IP

Como já tivemos oportunidade de ver, logo nas primeiras unidades, todos nós estamos integrados na maior rede de comunicação do mundo - a **Internet**. A Internet não só é uma rede, como também pode ser chamada de rede de redes, dada a sua elevadíssima complexidade e o aglomerado que sustém. Para organizar esta rede tem de existir uma **hierarquia**, a qual se torna cada vez mais próxima do utilizador real, quão mais baixo estivermos nela.

Estrutura hierárquica de redes

O primeiro patamar que atingimos na hierarquia de redes Internet é o que contém todos os fornecedores principais de Internet no mundo - a estes fornecedores de serviço Internet damos o nome de **ISP** (sigla inglesa para *Internet Service Provider*). Neste primeiro patamar, denominado de **Tier 1 ISP**, várias entidades veem-se como iguais, interligando-se de forma privada (**peers**). Esta relação está esquematicamente representada na Figura 8.1.

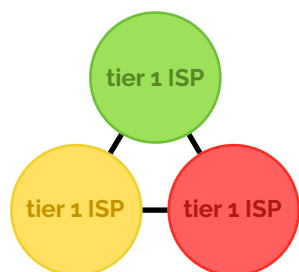


figura 8.1

Num segundo patamar temos fornecedores de serviço mais pequenos, estes, ligados a um ou mais ISP's do primeiro Tier e, possivelmente, a outros ISP's do mesmo nível. Esta relação está representada na Figura 8.2. Aqui, ISP's de nível 2 pagam aos ISP's do nível superior por conectividade com o resto da Internet, considerando-se clientes dos superiores. Se houver ligações entre ISP's do nível 2, estas serão privadas.

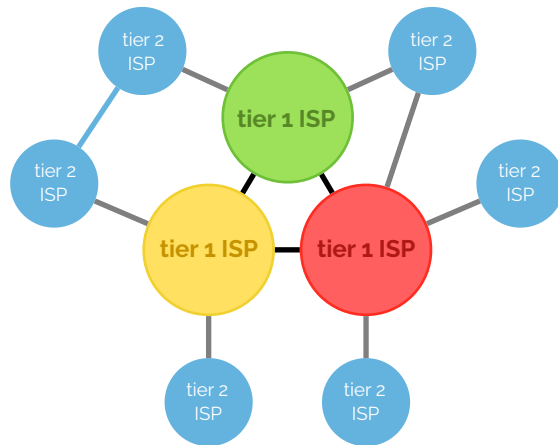


figura 8.2

Mais abaixo, ainda podemos encontrar fornecedores de serviço ainda mais pequenos, ainda mais próximos dos sistemas terminais (clientes finais) - são consideradas redes de acesso. Estes, claramente, são clientes dos ISP mais superiores (tier 2 e 1).

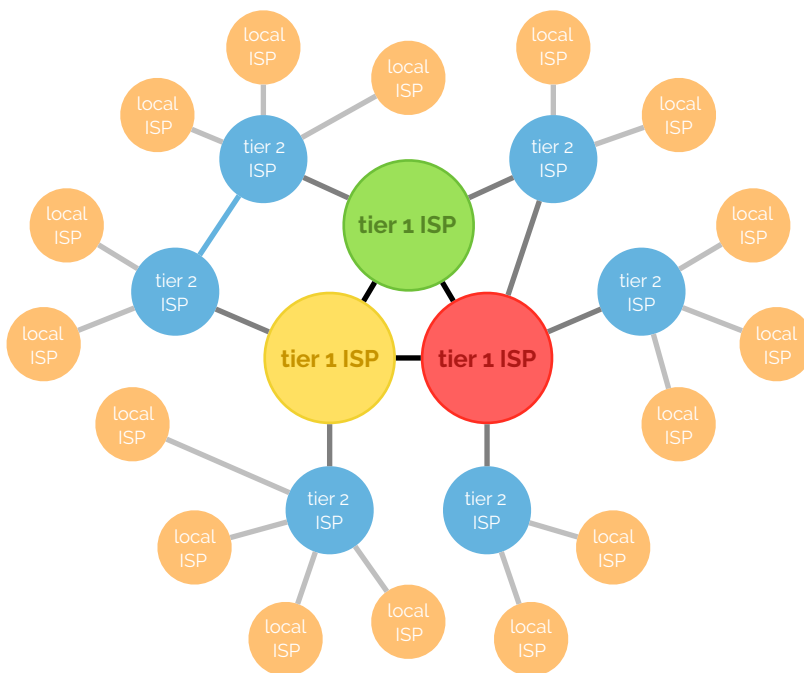


figura 8.3

Quando um determinado utilizador pretende enviar um pacote do seu terminal para outro terminal, o pacote viaja por uma dada sequência de redes até chegar ao seu destino. Mas como é que, na rede, se consegue saber qual é o caminho certo de uma dada origem para um destino? Dentro dos paradigmas das redes existe um conceito denominado de **sistemas autónomos** (AS - de *Autonomous System*) que são conjuntos de routers com uma política de encaminhamento própria, sob a responsabilidade de uma única administração. Cada sistema autónomo é identificado por um endereço único de 16 bits atribuído por um *Internet Registry* ou por um ISP. O encaminhamento que é feito dentro os sistemas autónomos é realizado por **IGP's** (sigla de *Interior Gateway Protocol*), tais como o RIP, OSPF e o IS-IS. Por outro lado, o encaminhamento entre sistemas autónomos é realizado por **EGP's** (sigla de *Exterior Gateway Protocol*) tais como BGP.

sistemas autónomos

IGP

EGP

Os IGP's e EGP's têm diferentes objetivos: por um lado, os IGP's focam-se em otimizar o desempenho nos encaminhamentos (*shortest path routing*); por outro lado, os EGP's têm como objetivo atender a questões de ordem política, económica e de segurança

(para além do desempenho). Nesta disciplina ir-nos-emos focar nas questões colocadas aos IGP's, principalmente no que toca ao encaminhamento RIP.

Distance vector versus link state

O encaminhamento ao nível do IP é feito através de percursos de custo mínimo (*shortest paths*). Para tal existem duas abordagens distintas à resolução deste problema, de captar o custo mínimo - distance vectors e link state. Os protocolos de **distance vector** têm como característica serem distribuídos, recorrendo a algoritmos como o Bellman-Ford (versão distribuída e assíncrona) para determinar os percursos mínimos. Alguns exemplos de protocolos que usam distance vectors são o RIP (que vamos estudar de seguida), o IGRP ou o EIGRP. Por outro lado, os protocolos de **link state** têm como característica principal fazer com que os routers mantenham uma base de dados com a topologia completa da rede, recorrendo a algoritmos de encaminhamento centralizados, como o algoritmo de Dijkstra, para determinar os percursos mínimos.

distance vector

link state

© Edsger Dijkstra

Protocolos distance vector

Como já vimos atrás, pela Equação de Bellman, temos que o comprimento do percurso mínimo de um nó para o nó inicial é igual à soma do comprimento do arco que une esse nó ao que se lhe segue no percurso mínimo ao comprimento do percurso mínimo desse nó para o nó inicial. Consideremos então o seguinte grafo da Figura 8.4 onde o custo de todas as ligações é unitário e os routers são ligados simultaneamente.

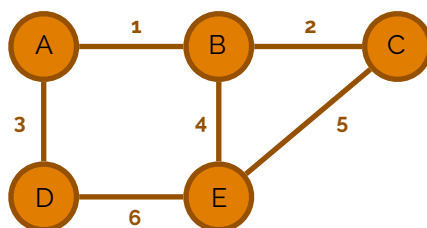


figura 8.4

Dado que estamos a estudar protocolos de distance vectors vejamos como é que a informação é preservada acerca dos routers e das distâncias entre eles e como é que eles partilham informação, pelo simples facto de se tratar de distance vectors.

Assim, na Figura 8.5 podemos ver as informações iniciais de todos os routers.

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A			A			A			A		
B			B	local	0	B			B			B		
C			C			C	local	0	C			C		
D			D			D			D	local	0	D		
E			E			E			E			E	local	0

figura 8.5

Então A difunde o vetor que contém ($A = 0$) nas suas ligações 1 e 3 o que produz o efeito visível em Figura 8.6.

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	1	A			A	3	1	A		
B			B	local	0	B			B			B		
C			C			C	local	0	C			C		
D			D			D			D	local	0	D		
E			E			E			E			E	local	0

figura 8.6

De A para B, via 1, agora é a vez de B difundir o seu vetor ($A = 1$, $B = 0$) nas suas ligações 1, 2 e 4, produzindo o resultado visível na Figura 8.7.

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	1	A	2	2	A	3	1	A	4	2
B	1	1	B	local	0	B	2	1	B			B	4	1
C			C			C	local	0	C			C		
D			D			D			D	local	0	D		
E			E			E			E			E	local	0

figura 8.7

Agora D difunde o seu vetor ($A = 1$, $D = 0$) nas suas ligações 3 e 6 (Figura 8.8).

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	1	A	2	2	A	3	1	A	4	2
B	1	1	B	local	0	B	2	1	B			B	4	1
C			C			C	local	0	C			C		
D	3	1	D			D			D	local	0	D	6	1
E			E			E			E			E	local	0

figura 8.8

Em simultâneo, A difunde o seu vetor ($A = 0$, $B = 1$, $D = 1$) nas ligações 1 e 3, C difunde ($A = 2$, $B = 1$, $C = 0$) nas ligações 2 e 5 e E difunde o vetor ($A = 2$, $B = 1$, $D = 1$, $E = 0$) nas ligações 4, 5 e 6 (Figura 8.9).

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	1	A	2	2	A	3	1	A	4	2
B	1	1	B	local	0	B	2	1	B	3	2	B	4	1
C			C	2	1	C	local	0	C			C	5	1
D	3	1	D	1	2	D	5	2	D	local	0	D	6	1
E			E	4	1	E	5	1	E	6	1	E	local	0

figura 8.9

As informações em falta podem ser garantidas pelas próximas partilhas, estas, provenientes de B, que difunde o vetor ($A = 1$, $B = 0$, $C = 1$, $D = 2$, $E = 1$) nas ligações 1, 2 e 4; de D, que difunde o vetor ($A = 1$, $B = 2$, $D = 0$, $E = 1$) nas ligações 3 e 6; e de E, que difunde o vetor ($A = 2$, $B = 1$, $C = 1$, $D = 1$, $E = 0$) nas ligações 4, 5 e 6 (Figura 8.10).

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	1	A	2	2	A	3	1	A	4	2
B	1	1	B	local	0	B	2	1	B	3	2	B	4	1
C	1	2	C	2	1	C	local	0	C	6	2	C	5	1
D	3	1	D	1	2	D	5	2	D	local	0	D	6	1
E	1	2	E	4	1	E	5	1	E	6	1	E	local	0

figura 8.10

Tendo toda a informação necessária para encaminhamento, podemos dizer que o algoritmo convergiu e que as tabelas de encaminhamento estão completas. Reparemos que no decorrer desta atualização de informação na tabela de encaminhamento, nem todas as informações dos vetores são atualizadas, isto porque nem sempre os dados novos são melhores que os que já estão na tabela. Esta questão e a ordem de difusão dos vetores são dois aspetos que são produto da aplicação de algoritmos como o de Bellman-Ford.

Consideremos agora que os routers detetam que a ligação 1 foi interrompida por alguma razão, o que é que acontece? Vejamos na Figura 8.11 como é que estes routers reagem.

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	∞	A	2	2	A	3	1	A	4	2
B	1	∞	B	local	0	B	2	1	B	3	2	B	4	1
C	1	∞	C	2	1	C	local	0	C	6	2	C	5	1
D	3	1	D	1	∞	D	5	2	D	local	0	D	6	1
E	1	∞	E	4	1	E	5	1	E	6	1	E	local	0

figura 8.11

Como as ligações foram interrompidas, a percepção de custo para os routers é que é muito alta - infinito (∞). Agora, há que ser repetida a atualização dos custos, até que a tabela convirja. Assim, A difunde o vetor ($A = 0, B = \infty, C = \infty, D = 1, E = \infty$) na ligação 3 e B difunde o vetor ($A = \infty, B = 0, C = 1, D = \infty, E = 1$) nas ligações 2 e 4 (Figura 8.12).

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	∞	A	2	∞	A	3	1	A	4	∞
B	1	∞	B	local	0	B	2	1	B	3	∞	B	4	1
C	1	∞	C	2	1	C	local	0	C	6	2	C	5	1
D	3	1	D	1	∞	D	5	2	D	local	0	D	6	1
E	1	∞	E	4	1	E	5	1	E	6	1	E	local	0

figura 8.12

De seguida C deve difundir o seu vetor ($A = \infty, B = 1, C = 0, D = 2, E = 1$) nas ligações 2 e 5; D difunde o seu vetor ($A = 1, B = \infty, C = 2, D = 0, E = 1$) nas ligações 3 e 6; E difunde o vetor ($A = \infty, B = 1, C = 1, D = 1, E = 0$) nas ligações 4, 5 e 6 (Figura 8.13).

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	1	∞	A	2	∞	A	3	1	A	6	2
B	1	∞	B	local	0	B	2	1	B	6	2	B	4	1
C	3	3	C	2	1	C	local	0	C	6	2	C	5	1
D	3	1	D	4	2	D	5	2	D	local	0	D	6	1
E	3	2	E	4	1	E	5	1	E	6	1	E	local	0

figura 8.13

Finalmente, para atualizar os últimos custos a infinitos para valores contáveis, A difunde o vetor ($A = 0, B = \infty, C = 3, D = 1, E = 2$) nas ligações 2 e 4, D difunde o vetor ($A = 1, B = 2, C = 2, D = 0, E = 1$) nas ligações 3 e 6 e E difunde o vetor ($A = 2, B = 1, C = 1, D = 1, E = 0$) nas ligações 4, 5 e 6 (Figura 8.14).

A para...	via	custo	B para...	via	custo	C para...	via	custo	D para...	via	custo	E para...	via	custo
A	local	0	A	4	3	A	5	3	A	3	1	A	6	2
B	3	3	B	local	0	B	2	1	B	6	2	B	4	1
C	3	3	C	2	1	C	local	0	C	6	2	C	5	1
D	3	1	D	4	2	D	5	2	D	local	0	D	6	1
E	3	2	E	4	1	E	5	1	E	6	1	E	local	0

figura 8.14

Neste momento o algoritmo voltou a convergir e as tabelas estão de novo completas.

Problema da contagem para infinito e a separação de horizontes

Consideremos o seguinte grafo (rede) onde todas as ligações são bidirecionais e onde os custos de todas as ligações são iguais a 1, à exceção da ligação entre 1 e 4, com o custo de 10 (Figura 8.15). Consideremos também que as condições iniciais correspondem aos comprimentos dos percursos mínimos dos nós 2, 3 e 4 para o nó 1 ($D_2 = 1, D_3 = 2$ e $D_4 = 2$).

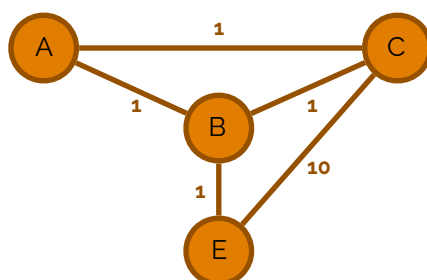


figura 8.15

Admitamos agora que após o instante inicial a ligação de 1 a 2 falha, considerando o algoritmo síncrono (isto é, que todas as trocas de mensagem entre nós são simultâneas) (Figura 8.16).

tempo	t = 0	t = 1	t = 2	t = 3	...	t = 7	t = 8	t = 9	t = 10
(C, PS)	D ₂ = 1, 1	3, 4	4, 4	5, 4		9, 4	10, 4	11, 4	11, 4
custo,	D ₃ = 2, 2	3, 4	4, 4	5, 4		9, 4	10, 4	11, 4	11, 4
próximo	D ₄ = 2, 2	3, 3	4, 3	5, 3		9, 3	10, 3	10, 1	10, 1
salto									

figura 8.16

Neste caso que analisámos o algoritmo acabou por convergir ao fim de 10 iterações. No entanto, quando o nó destino fica isolado de uma parte da rede, o número de iterações necessárias é **infinito**. Para resolver este tipo de situações criou-se uma técnica chamada de **separação de horizontes** (em inglês *split horizon*).

Esta técnica de separação de horizontes baseia-se no seguinte: se um nó A encaminha pacotes para um nó destino D através de um seu nó vizinho B, não faz sentido que o nó B tente encaminhar pacotes para o nó destino D através do nó A, pois não é necessário o nó A anunciar na ligação entre os nós A e B qual o custo para o nó destino D. Basicamente, cada router anuncia numa sua interface 0, os destinos de cada uma das outras interfaces, pelo que o anúncio em cada interface é diferente (daí separação de horizontes). Mas há uma pequena variância na utilização de separação de horizontes - se esta técnica incluir **poisonous reverse**, isto significa que também pode haver o anúncio de infinito e, caso contrário, não há anúncio.

infinito

separação de horizontes

poisonous reverse

Routing Information Protocol (RIP)

Um dos protocolos IGP distance vectors é o **RIP** (acrónimo de *Routing Information Protocol*). Neste protocolo, cada router mantém um vetor distância constituído por uma lista das redes IP que conhece e, para cada rede, da sua estimativa atual do melhor custo. Estas informações são anunciadas regularmente pelos routers aos seus vizinhos, utilizando essa informação para atualizar os seus vetores distância.

Neste protocolo o custo de um percurso de um router para uma rede é dado pelo número de saltos entre outros routers intermédios, considerando como custo máximo o valor 15 (considerando também o valor 16 como infinito).

Cada router determina as entradas da sua tabela com base nos vetores distância recebidos dos vizinhos - para cada rede existente, o router inclui na sua tabela de encaminhamento uma entrada para cada um dos seus vizinhos que lhe proporcionem um percurso de custo mínimo, embora, no caso em que há mais que um vizinho para a mesma rede destino, o router deve ser capaz de fazer **load balance** do tráfego, isto é, encaminhar em igual percentagem os pacotes pelos diferentes vizinhos. Consideremos a seguinte rede da Figura 8.17 e vejamos as tabelas de encaminhamento de cada um dos quatro routers.

As mensagens trocadas entre os vários routers de uma rede dependem também da versão protocolar do RIP. Numa primeira versão, a **RIPv1**, definida no RFC 1059 do IETF, existem dois tipos de mensagens: um RIP Request e um RIP Response. Por norma, num funcionamento regular apenas haverá mensagens **RIP Response** a navegar na rede, entre routers - é esta a mensagem que contém o vetor distância e que é enviada periodicamente para a rede (no máximo de 30 em 30 segundos) ou quando a informação é alterada (**triggered updates**) - estas mensagens são enviadas com destino broadcast⁴. Opcionalmente, pode dar-se o caso de um router entrar numa rede e efetuar um **RIP Request** - nestas mensagens podem-se pedir, então, informações acerca de um destino em específico ou mesmo até sobre todos os destinos, resposta a qual, virá sobre a forma de um

RIP

load balance

RIPv1

< RFC 1059

RIP Response

triggered updates

RIP Request

⁴ As mensagens RIP Response quando são enviadas para o endereço broadcast não são acessíveis por todos os equipamentos presentes na rede, mas apenas por aqueles que se encontram na rede e se encontram à escuta no porto 520 (exclusivo a Routers), pelo que as comunicações entre routers são garantidas por UDP, da camada 4 do OSI, assunto que iremos estudar mais à frente.

RIP Response direcionado exclusivamente para o endereço de quem efetuou o RIP Request.

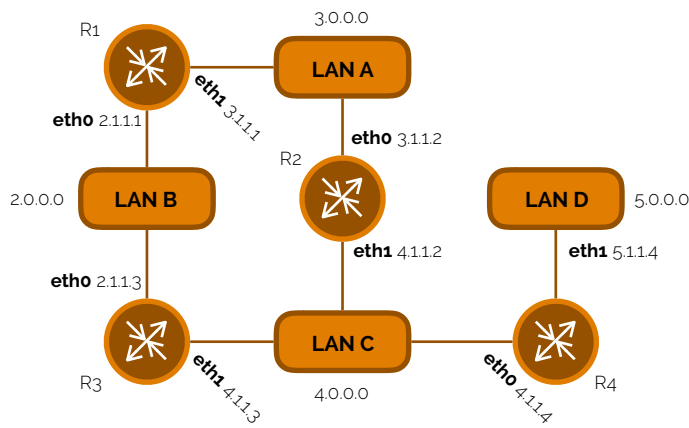


figura 8.17

Router 1

```
C 2.0.0.0/8 is directly connected, eth0
C 3.0.0.0/8 is directly connected, eth1
R 4.0.0.0/8 [120/1] via 3.1.1.2, 00:16:00, eth1
  [120/1] via 2.1.1.3, 00:00:12, eth0
R 5.0.0.0/8 [120/2] via 3.1.1.2, 00:00:13, eth1
  [120/2] via 2.1.1.3, 00:00:02, eth0
```

Router 2

```
R 2.0.0.0/8 [120/1] via 4.1.1.3, 00:00:26, eth1
  [120/1] via 3.1.1.1, 00:00:02, eth0
C 3.0.0.0/8 is directly connected, eth0
C 4.0.0.0/8 is directly connected, eth1
R 5.0.0.0/8 [120/1] via 4.1.1.4, 00:00:23, eth1
```

Router 3

```
C 2.0.0.0/8 is directly connected, eth0
R 3.0.0.0/8 [120/1] via 4.1.1.2, 00:00:06, eth1
  [120/1] via 2.1.1.1, 00:00:05, eth0
C 4.0.0.0/8 is directly connected, eth1
R 5.0.0.0/8 [120/1] via 4.1.1.4, 00:00:23, eth1
```

Router 4

```
R 2.0.0.0/8 [120/1] via 4.1.1.3, 00:00:13, eth0
R 3.0.0.0/8 [120/1] via 4.1.1.2, 00:00:06, eth0
C 4.0.0.0/8 is directly connected, eth1
C 5.0.0.0/8 is directly connected, eth1
```

Até agora, se repararmos bem na Figura 8.17, o anúncio dos vetores distância é feito de forma a que cada router anuncie o vetor distância completo por todas as interfaces. Como já vimos antes trata-se de uma situação redundante e algo perigosa para o correto funcionamento do encaminhamento na rede. Para corrigir esta situação, podemos ativar o *split horizon*, o que fará com que, em cada interface, o router anuncie apenas as redes destino para as quais essa interface não é usada no encaminhamento dos pacotes de dados. Na Figura 8.18 podemos ver a diferença da desativação/ativação da técnica de *split horizon* nos pacotes RIP enviados pelo Router 1 para a rede 3.0.0.0 da Figura 8.17.

IP Routing Information Protocol

```
Command: 2 (Reponse)
Version: 1
Address Family ID: 2
  IP Address: 2.0.0.0
  Metric: 1
Address Family ID: 2
  IP Address: 3.0.0.0
  Metric: 1
Address Family ID: 2
  IP Address: 4.0.0.0
  Metric: 2
Address Family ID: 2
  IP Address: 5.0.0.0
  Metric: 3
```

sem split horizon

IP Routing Information Protocol

```
Command: 2 (Reponse)
Version: 1
Address Family ID: 2
  IP Address: 2.0.0.0
  Metric: 1
```

com split horizon

figura 8.18

RIP com/sem split horizon

Se houver uma falha de ligação, em pleno curso de execução, o RIP imediatamente executa uma mensagem de RIP Request para aferir as alterações que foram feitas. Consideremos assim o caso em que a ligação entre a LAN D e o Router 4 fica inativa, por uma determinada razão. Observemos assim 3 capturas seguidas com a diferença de 30 segundos entre o primeiro e o terceiro pacote (Figura 8.19) - os seguintes pacotes são provenientes de 4.1.1.4.

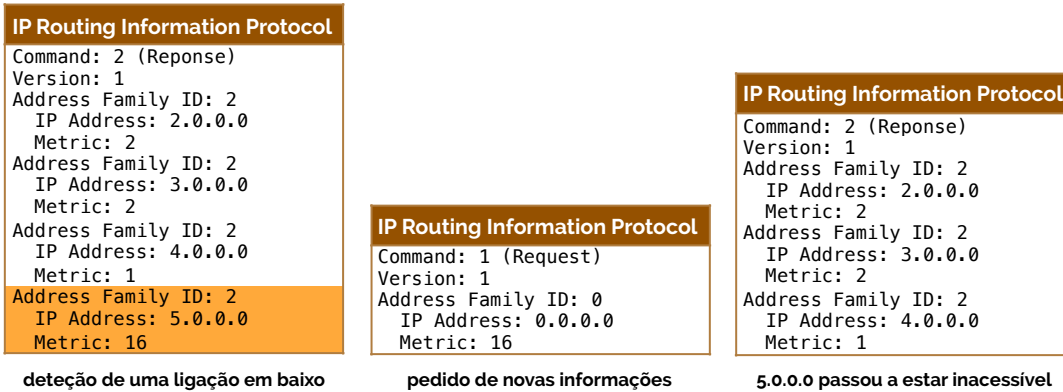


figura 8.19

Este protocolo de encaminhamento IP, ainda assim, tem um grande problema - não é aplicável, por exemplo, a sub-redes. Para tal, foi criada uma nova versão deste protocolo, de seu nome **RIPv2**. Este protocolo está assim definido no RFC 2453 do IETF. Algumas das diferenças desta versão com a anterior são a capacidade de fornecer autenticação no acesso a determinados caminhos (permite evitar que uma estação envie mensagens RIP que modifiquem as tabelas de encaminhamento), o suporte de encaminhamento em sub-redes (suporta **VLSM** - sigla inglesa para *Variable Length Subnet Masking*) e envio de mensagens RIP para o endereço multicast 224.0.0.9, ao invés do de broadcast (assim só os routers que pertencem ao grupo é que recebem e processam as mensagens).

RIPv2
[RFC 2453](#)

O RIPv2 não liga a classes de endereços IP, pelo que as suas mensagens incluem as máscaras de rede, suportando máscaras de rede dos mais variados tamanhos. Este protocolo automaticamente sumaria as rotas em redes sem limitações de classes, embora esta capacidade possa ser desativada.

VLSM

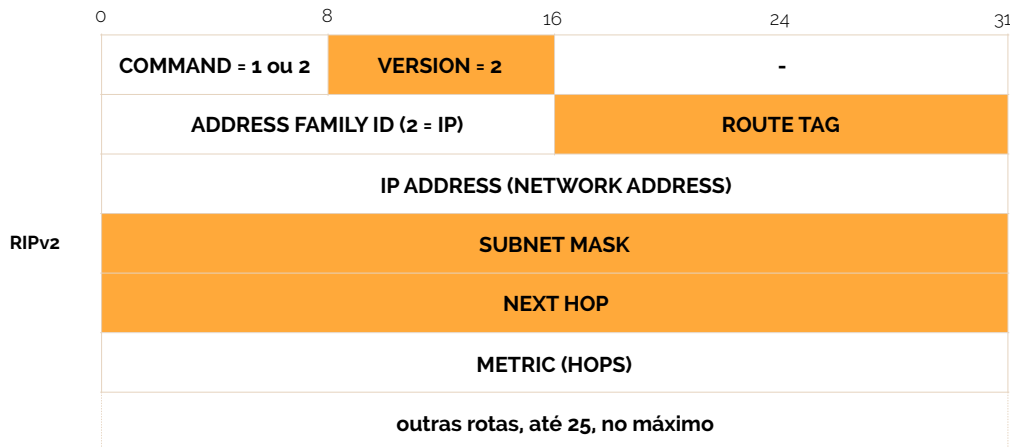
As mensagens RIPv2 possuem, então, novos campos, como a máscara de rede, uma **route tag** - atributo designado a uma rota que deve ser preservado e re-anunciado com uma rota e que fornece um método de separação de rotas para redes inseridas no domínio de encaminhamento RIP entre outras rotas que podem ter sido importadas de outros protocolos (redistribuição) e **next hop** - endereço IP para onde os pacotes desta entrada devem ser reencaminhados e, cujo valor, se for 0.0.0.0, indica que deve haver um reencaminhamento para o router que enviou a presente mensagem. Na Figura 8.20 podemos ver ambos formatos de mensagem RIPv1 e RIPv2.

route tag

next hop

	0	8	16	24	31
RIPv1	COMMAND = 1 ou 2		VERSION = 1		-
	ADDRESS FAMILY ID (2 = IP)			-	
	IP ADDRESS (NETWORK ADDRESS)				
	-				
	-				
	METRIC (HOPS)				
	outras rotas, até 25, no máximo				

figura 8.20
formato RIPv1

figura 8.21
formato RIPv2

Mas esta versão do RIP não suporta uma outra característica que já estudamos antes - endereçamento IPv6 -, para tal criou-se a versão **RIPng** (*ng* de *new generation*). Esta nova versão do RIP é equivalente ao RIPv2, só que para endereçamento IPv6.

RIPng

9. Camada de Transporte

Até aqui temos vindo a estudar a segunda e terceira camada protocolar OSI, sobre a qual conseguimos agora perceber como é que se processam as transferências de dados e a organização de uma rede, tal como é que os pacotes podem ser encaminhados. Agora vamos subir um pouco mais, chegando à **camada de transporte**, a qual se encarrega de entregar os pacotes de forma segura/rápida aos destinos traçados. A camada de transporte, em inglês *transport layer*, é então a fornecedora da lógica de comunicação entre as aplicações usadas e em execução entre dois terminais. Nesta lógica formam-se dois lados: um lado - o lado **emissor** - divide a informação em numerosas partes e envia tais segmentos para a camada de rede; o outro lado - o lado **recetor** - monta todos os segmentos em informação e passa à camada de aplicação (camada acima da de transporte). Para que isto aconteça, tal como todas as ações que são elaboradas no meio de computação, há que existir protocolos, e aqui iremos falar de dois: o TCP e o UDP.

camada de transporte

emissor
recetor

Embora ainda seja cedo referir a camada de aplicação, é importante, desde já, ter uma noção em mente: a noção de **porto**. Para aprendermos o que é um porto, consideremos as docas de uma determinada cidade como o nosso computador. Quando há uma importação de uma cidade para a nossa, a entidade responsável por ir buscar a carga deve saber onde, nas docas, é que tem de a buscar - da mesma forma que quem vem no navio para a descarregar também não pode descarregar num sítio qualquer. O mesmo acontece no nosso computador - quando há uma comunicação entre aplicações, esta é feita num local em específico, este, que ambas as partes que lhe têm acesso, precisam de conhecer. A este local damos o nome de porto. Antes já referimos este termo, por exemplo, quando estudámos DHCP, no qual indicámos que se usavam os portos 67 (DHCP Server) e 68 (DHCP Client). A comunicação com estes portos (a ponte entre o navio e a doca) é denominada de **socket** (como uma tomada de eletricidade, que tem um encaixe perfeito com o cabo que nela inserimos).

porto

socket

User Datagram Protocol (UDP)

O primeiro protocolo de transporte que vamos estudar é o **UDP** (sigla inglesa de *User Datagram Protocol*). Este protocolo fornece um serviço com as mesmas capacidades de transporte de dados que o IP fornece na sua camada, permitindo a troca de dados entre aplicações através de um cabeçalho e um identificador de porto. Assim sendo, há a possibilidade de entregar um determinado conjunto de dados a mais que um destino. O datagrama UDP tem então a forma representada na Figura 9.1.

UDP

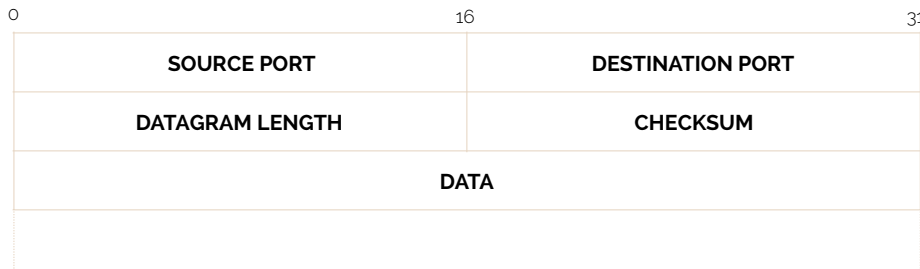


figura 9.1
datagrama UDP

O UDP, por se basear inteiramente no processamento protocolar do IP é um protocolo que não garante qualidade na transmissão, isto, em termos de correção e completude na entrega dos pacotes. Isto acontece porque, na transmissão, se alguma parte se perder não há forma de recuperação.

Transmission Control Protocol (TCP)

O segundo e último protocolo de transporte que falamos nesta disciplina é o **TCP** (sigla inglesa para *Transmission Control Protocol*). Este protocolo difere bastante do UDP, principalmente em termos de complexidade. Sendo aplicado em comunicações ponto-a-ponto (isto é, tendo um emissor e um recetor, unicamente) tem a capacidade de garantir a correção (se perder informação, retransmite) e avaliar o fluxo de informação nos variados canais por onde atravessa, modificando as taxas de transferência. Também usa canais full-duplex e orientada à ligação - diz-se que é orientada à ligação porque toma um esquema de **handshaking** entre emissor e recetor.

TCP

handshaking

Vejamos então como é que é a estrutura de um segmento TCP (Figura 9.2).

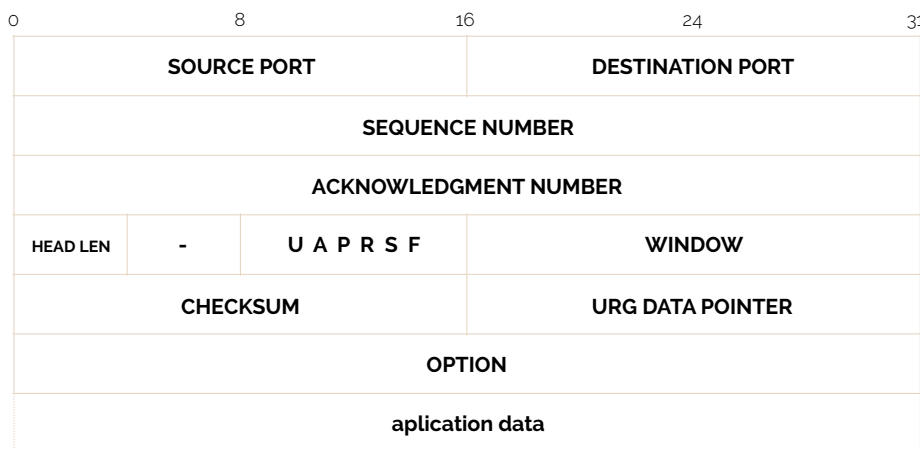


figura 9.2
segmento TCP

No segmento TCP existem vários campos que carecem de explicação:

- **SOURCE PORT** - porta de origem da transmissão;
- **DESTINATION PORT** - porta de destino da transmissão;
- **SEQUENCE NUMBER** - número de sequência da transmissão, também abreviada de SN;
- **ACKNOWLEDGMENT NUMBER** - número de reconhecimento da transmissão, também abreviada de ACK;
- **HEADER LENGTH** - tamanho do cabeçalho de transporte;
- **Flag U** - embora não seja muito usada, significa que o transporte da mensagem que comporta é urgente;
- **Flag A** - flag que permite averiguar se o número de reconhecimento (ACK) é válido;

- **Flag P** - embora não muito usada, esta flag permite informar que há a necessidade de efetuar um *push* imediato;
- **Flag R** - flag que indica a necessidade de restaurar a ligação (reset);
- **Flag S** - normalmente abreviada de SYN, esta flag permite explicitar a vontade de iniciar uma ligação;
- **Flag F** - normalmente abreviada de FIN, esta flag permite explicitar a vontade de terminar uma ligação;
- **WINDOW** - número de bytes que o recetor tem capacidade de aceitar, de momento;
- **CHECKSUM** - campo equivalente ao do UDP para deteção e correção de erros;
- **URG DATA POINTER** - campo que indica início de excerto classificado como urgente, nos dados que comporta.

Como vimos pela descrição dos campos do segmento TCP, este protocolo utiliza as designações de número de sequência e número de reconhecimento. Assim, como número de sequência (SN) considera-se o primeiro byte dos dados do segmento e, como número de reconhecimento (ACK), considera-se o número de sequência do byte seguinte esperado pelo outro interveniente da transmissão, sendo um número cumulativo. Vejamos, assim, uma comunicação (já estabelecida) entre um computador A e um computador B, por TCP, na Figura 9.3.

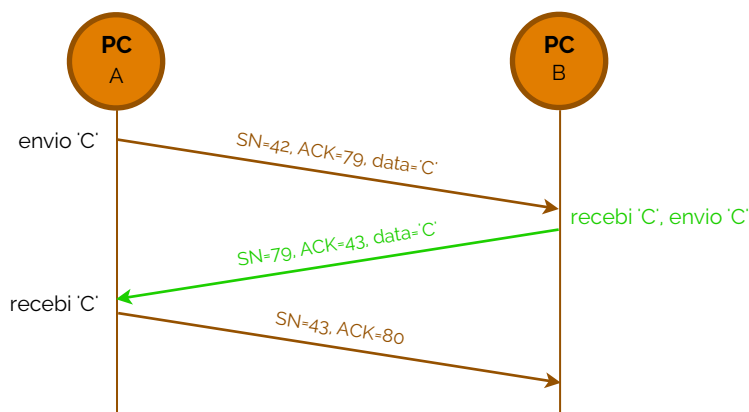


figura 9.3

Na Figura 9.3 podemos então ver uma comunicação já estabelecida onde o utilizador do computador A envia um carater 'C' para B, transmissão essa que tem o número de sequência 42 e que, por si, confirma uma transmissão anterior (não visível na figura) com número de sequência 78, isto, porque o ACK é de 79 (SN anterior + 1). Chegando ao computador B, este envia uma mensagem onde reconhece a chegada do carater 'C' (ACK = 43) e envia novo carater 'C', agora para A, numa mesma transmissão, com número de sequência 79. Quando chega a A, este envia uma nova mensagem, rotulada com número de sequência 43 onde leva o reconhecimento (ACK) 80.

Mas como é que se inicia uma transmissão TCP? Aquando da descrição dos campos do segmento TCP, verificámos que existem flags que indicam a vontade de uma das partes de iniciar uma ligação (flag SYN). Do mesmo modo, também verificámos que o envio de um ACK também é feito através da flag em conjunto com o valor do campo ACKNOWLEDGMENT NUMBER. Agora é só montar as peças: inicialmente vimos que o TCP usa a técnica de handshake, logo, para iniciar a ligação, primeiramente o cliente deve enviar um segmento TCP com a flag SYN ativa, o qual deve ser respondido através de um novo segmento TCP, com a mesma flag ativa, juntamente com a flag ACK, seguido de um novo segmento do cliente para o servidor, a reconhecer o (SYN, ACK) do servidor, enviando um ACK apenas. Basicamente, o **início de ligação** TCP usa um padrão entre cliente servidor, de handshake, com a ordem (SYN), (SYN, ACK), (ACK). A partir deste momento a ligação está estabelecida - dá-se o nome de **three-way handshake**.

início de ligação

three-way handshake

Para terminar a ligação, o processo é algo análogo: primeiramente, quem pretende terminar a ligação envia um segmento onde inclui a flag FIN, que do outro lado notifica a aplicação do terminus da ligação, enviando de volta um segmento com (FIN, ACK) e respondendo de volta com ACK. Em suma, o **fim de ligação TCP** segue um padrão entre duas partes, de **four-way handshake**, com a ordem (FIN), (ACK), (FIN), (ACK).

fim de ligação
four-way handshake

Já a transmissão de dados em TCP não se julga apenas pelo número de sequência e número de reconhecimento. Como pudemos verificar antes, o segmento TCP também possui um campo denominado WINDOW - **janela**. Esta janela é, como dito anteriormente, o número de bytes que o recetor tem capacidade de aceitar, de momento. Isto existe porque a ligação TCP é gerada entre dois buffers (o do emissor e o do recetor) que possuem um determinado tamanho. Para que a ligação possa ser eficiente, o buffer recetor só poderá receber mais informação sem perigo de a perder, se ainda possuir espaço suficiente para a preservar. Caso não tenha, a transmissão terá de estagnar até haver espaço. Na Figura 9.4 podemos ver um exemplo de comunicação TCP, desde a sua ligação, ao seu fim.

janela

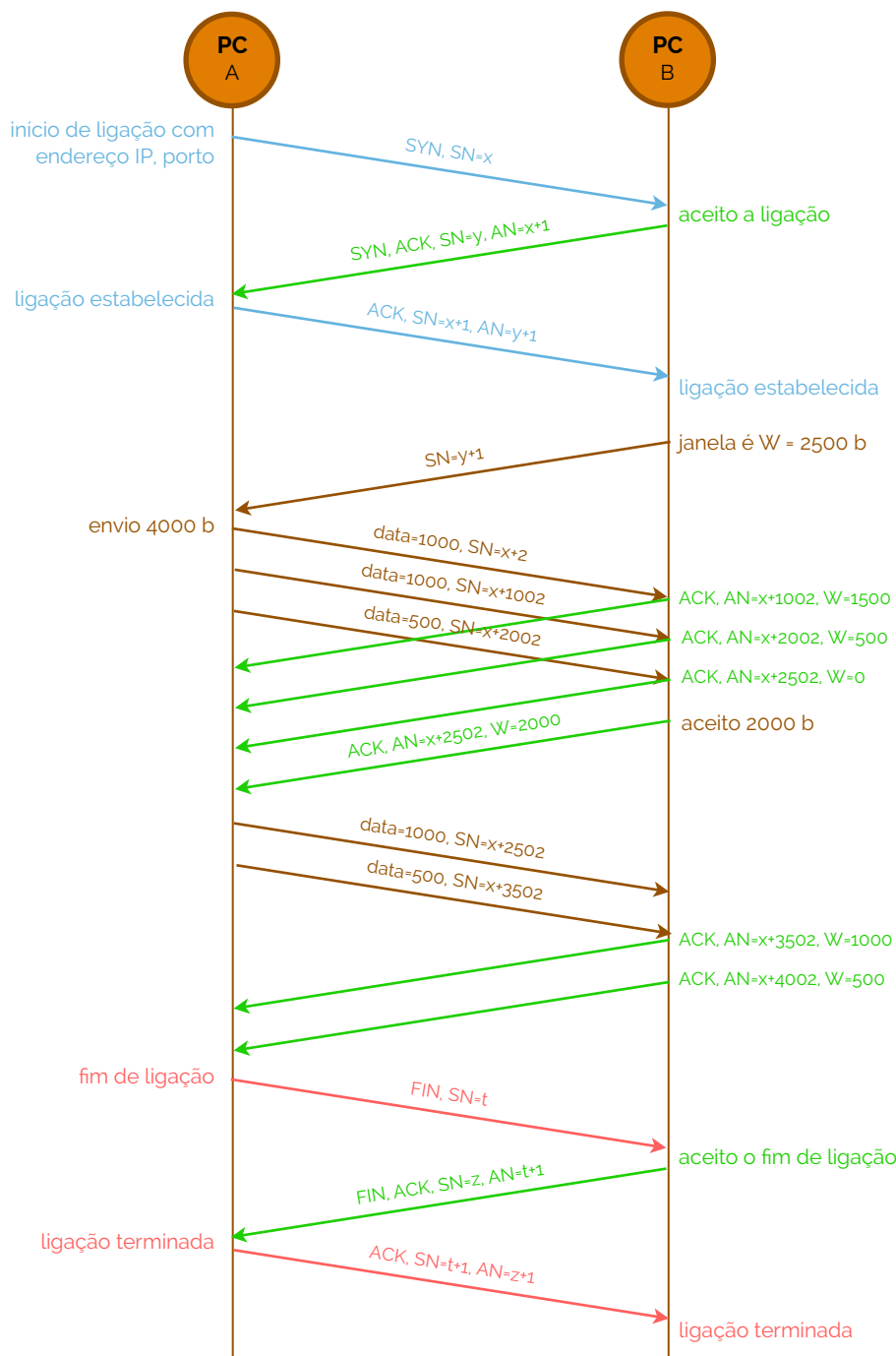


figura 9.4

Estimativas de RTT e timeout do TCP

Uma das grandes vantagens do TCP face ao UDP é que consegue reagir aos efeitos da falta de conectividade de uma determinada rede IP ou do congestionamento na mesma - recordemos que possui a capacidade de **controlo de fluxo**. Esta capacidade surge pelo facto de o protocolo abranger casos em que o tempo de retransmissão de pacotes (também denominado de timeout) tem de ser estimado através da manipulação do tempo de ida-e-volta (**round trip time** - RTT) ou em que a janela de bits de transmissão tem de sofrer alterações, aumentando ou diminuindo a taxa de transferência, dependendo das condições do canal de comunicação.

controlo de fluxo

round trip time

Embora o mecanismo de timeout/retransmissão pareça simples e linear, alguns dos seus aspetos mais subtis merecem especial atenção da nossa parte. Talvez a questão mais pertinente seja qual o tempo dos intervalos de timeout. Claramente, o timeout deve ser maior que o tempo de ida-e-volta (RTT) da ligação, isto é, o tempo desde que o segmento é enviado até que é reconhecido. De outra forma, seriam enviadas retransmissões completamente desnecessárias. Mas quão longo deve ser o timeout? Deverá ser usado um cronómetro associado com cada e todos os segmentos não reconhecidos?

Vejamos então como é que o TCP estima o tempo de ida-e-volta entre o emissor e o recetor. A amostra RTT (denominada aqui de `SampleRTT`) por um segmento é a quantidade de tempo entre o momento em que o segmento é enviado (isto é, passado para IP) e o momento em que é reconhecido pelo recetor. Ao invés de medir o `SampleRTT` por cada segmento transmitido, a maior parte das implementações de TCP usam apenas uma medida de `SampleRTT` de tempo em tempo. Isto é, num determinado instante de tempo, o `SampleRTT` não só é estimado para apenas um dos transmitidos, como também é estimado para os segmentos que não foram reconhecidos, levando a um novo valor de `SampleRTT`, aproximadamente por cada tempo de ida-e-volta. Mais, o TCP nunca recalcula um `SampleRTT` de uma retransmissão - só mede novos `SampleRTT` para segmentos que foram transmitidos uma vez.

Claramente, os valores de `SampleRTT` irão variar de segmento em segmento devido a possíveis diferenças de congestionamento nos routers e de carga variável nos sistemas terminais. Dada esta variância, qualquer valor de `SampleRTT` pode ser considerado como atípico. Para estimar um tempo de ida-e-volta típico, será então normal considerar a média de vários valores de `SampleRTT`. Assim, o TCP mantém uma média, denominada de `EstimatedRTT`, dos valores de `SampleRTT`. Após obter um novo valor de `SampleRTT`, o TCP atualiza o `EstimatedRTT` de acordo com a expressão de Equação 9.1.

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

equação 9.1

A fórmula da Equação 9.1 está escrita sob a forma programável, onde o valor de α deve ser considerado como uma constante e, segundo a norma RFC 2988, deverá ser de $1/8$, isto é, $\alpha = 0.125$ (Equação 9.2).

< RFC 2988

$$\text{EstimatedRTT} = 0.875 * \text{EstimatedRTT} + 0.125 * \text{SampleRTT}$$

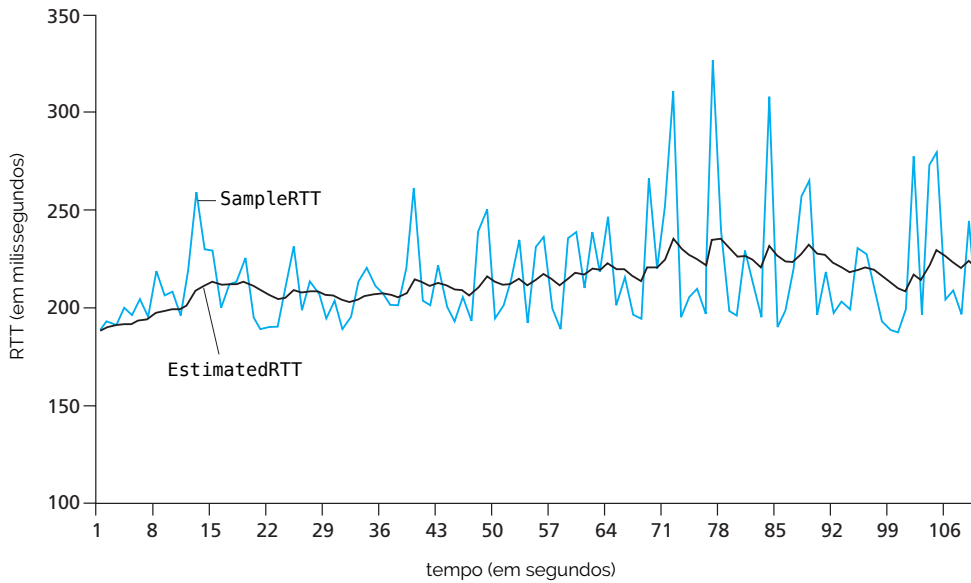
equação 9.2

Note-se que o `EstimatedRTT` é uma média ponderada dos valores de `SampleRTT`. Na Figura 9.5 podemos ver os valores de `SampleRTT` em comparação com os valores de `EstimatedRTT`, para o α de $1/8$, numa ligação TCP entre `gaia.cs.umass.edu` e `fantasia.eurecom.fr`. Claramente, as variações nos valores de `SampleRTT` são fortemente suavizadas no cálculo de `EstimatedRTT`.

TCP como um motor de transmissão fiável de dados

O TCP, como já tivemos oportunidade de verificar, cria um canal seguro de passagem de dados. Como também já devemos ter consciência, o IP não é fiável, isto porque não garante qualquer integridade na entrega de dados, muito menos a ordem de

figura 9.5



despacho. Com o serviço IP, os datagramas podem também facilmente consumir os buffers dos routers e nunca chegar a atingir o seu destino traçado, chegar fora de ordem ou até mesmo ficar corrompidos. O papel do TCP aqui é evitar estas más condições de "chegada" destes pacotes, sendo que garante-se que este protocolo seja capaz de levar os pacotes de um emissor a um recetor sem que estejam fora de ordem, corrompidos ou que os buffers se encham desmesuradamente. Mas como é que isto acontece? Nós iremos estudar duas perspetivas que se integram, no fim, uma na outra. Primeiramente iremos descrever como é que um emissor TCP simples funciona, que recupera os pacotes através dos timeouts; depois iremos apresentar um modelo mais completo onde são adicionados reconhecimentos em duplicado para deteção de falhas e as suas devidas correções. Nesta discussão, até ao fim desta unidade, iremos considerar que as comunicações são feitas numa só direção, do computador A para o computador B, e que o computador A está a tentar enviar um ficheiro de largas dimensões.

Um emissor TCP simples apenas se baseia na conceção de três eventos: a receção de um dado de uma aplicação, um timeout ou um reconhecimento de um outro dado. Sobre a ocorrência do primeiro evento, o TCP recebe os dados da aplicação, encapsula-os num segmento e passa-o para IP (note-se que cada segmento conta com um número de sequência e que o cronómetro ainda não está a correr para outro segmento, sendo que é iniciado apenas quando este é passado para IP).

Quanto à ocorrência do segundo evento - o de timeout - o TCP responde através da retransmissão direta do segmento que o causou, restaurando o cronómetro de seguida. Já perante o terceiro evento que deve ser processado pelo TCP, o de reconhecimento (ACK) de um segmento pelo recetor, o TCP deve comparar o valor y do ACK com a sua variável que contém o número de sequência do byte não reconhecido mais antigo que possui. Como já foi indicado anteriormente, o TCP usa reconhecimentos cumulativos, isto é, se se reconhece a chegada do ACK y , então reconhecem-se todos os anteriores. Na Figura 9.6 temos uma descrição de um emissor simples TCP.

Consideremos então um computador A que envia um segmento para o computador B - para tal, consideremos que este segmento tem número de sequência 92 e contém 8 bytes de dados. Depois de enviar este segmento, o computador A aguarda que o segmento seja recebido em B, mas o reconhecimento de B para A perde-se e o computador A retransmite o mesmo segmento. Claro que, quando o computador B recebe a retransmissão, pelo número de sequência, este repara que o segmento contém dados que já foram antes recebidos, pelo que irá ignorar o segmento. Este cenário está representado na Figura 9.7.

Num segundo cenário, um computador A envia dois segmentos de volta para trás - o primeiro segmento tem número de sequência 92 e 8 bytes de dados, e o segundo segmen-

```

NextSeqNum = InitialSeqNumber
SendBase = InitialSeqNumber

```

figura 9.6

```

para sempre alternar sobre evento {
  evento: dados recebidos de aplicação
    criar um segmento com número de sequência NextSeqNum
    se (cronómetro está parado)
      iniciar cronómetro
    passar segmento a IP
    NextSeqNum = NextSeqNum + tamanho(dados)
    break;
  evento: timeout
    retransmitir segmento não reconhecido com número de sequência menor
    iniciar cronómetro
    break;
  evento: reconhecimento recebido, com campo ACK igual a y
    se (y > SendBase) {
      SendBase = y
      se (existe algum segmento não reconhecido)
        iniciar cronómetro
    }
    break;
}

```

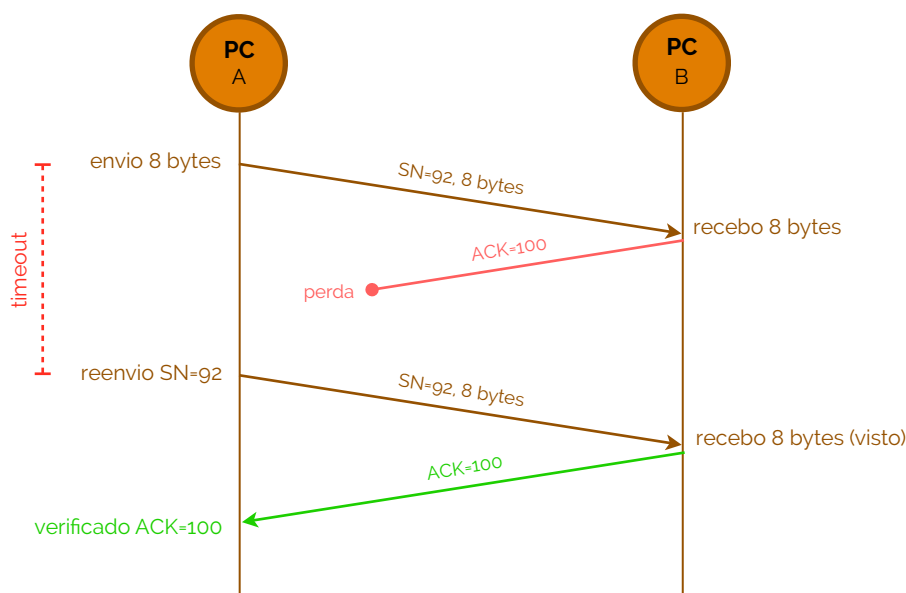


figura 9.7

to tem número de sequência 100 e 20 bytes de dados. Suponhamos agora, que ambos os segmentos chegam intactos a B e este encaminha reconhecimentos de ambos para A - o primeiro com número 100 e o segundo com número 120 -, mas nenhum dos dois reconhecimentos chega a A antes do timeout. Quando o timeout ocorre, o computador A reenvia o primeiro segmento com número de sequência 92 e reinicia o cronómetro. Até que o ACK do segundo segmento chegue antes do timeout, o segundo segmento não será retransmitido. Este cenário está representado na Figura 9.8.

Um último cenário mostra a capacidade dos reconhecimentos cumulativos (como o cenário anterior também o mostrava). Suponhamos para tal que o computador A envia dois segmentos, tal como no exemplo anterior. O reconhecimento do primeiro é perdido na rede e, mesmo antes do evento de timeout, o computador A recebe o reconhecimento do segundo segmento (ACK = 120). Assim sendo, como os reconhecimentos são cumulativos, se o B recebeu 120, então também recebeu tudo o que for menor que 120, logo também recebeu 100, pelo que o computador A não terá de reenviar nenhum dos segmentos. Este cenário está representado na Figura 9.9.

Um dos problemas das retransmissões ativadas por timeout é o facto do período de timeout poder ser relativamente longo. Quando um segmento é perdido, este tempo de timeout, por ser longo demais, força o emissor a atrasar o reencaminhamento do pacote perdido, consequentemente, aumentando o atraso de terminal a terminal. Felizmente, o emissor pode frequentemente detetar a perda de pacotes muito antes do timeout ocorrer, notando se existe **duplicação de ACK's**. Uma duplicação de um ACK (reconhecimento em

duplicação de ACK

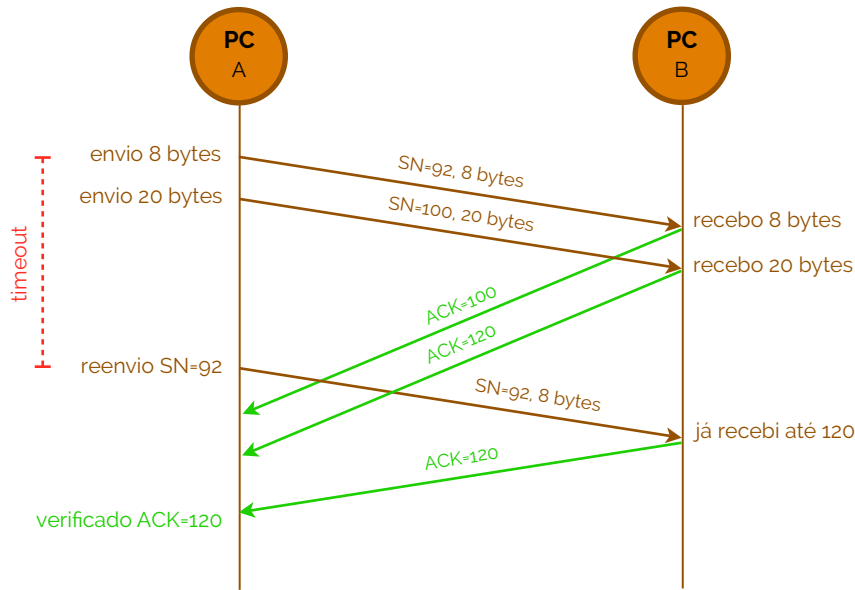


figura 9.8

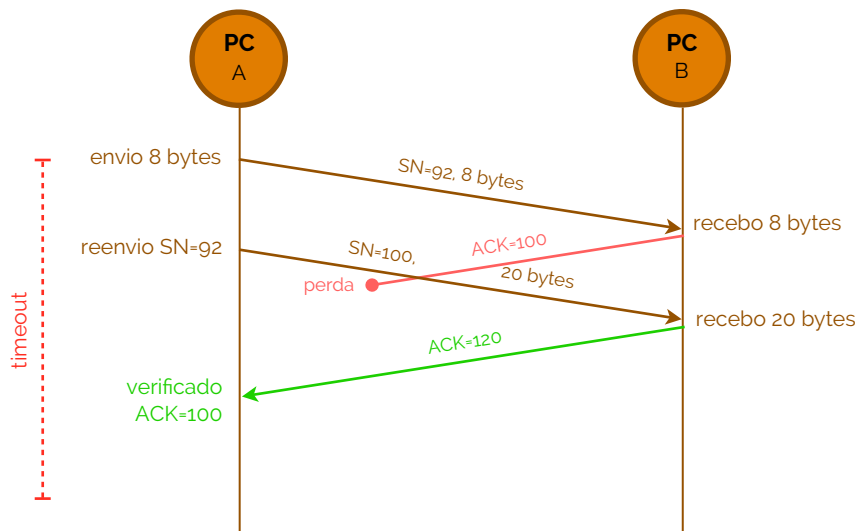


figura 9.9

duplicado) é um reconhecimento de um segmento que o emissor já recebeu anteriormente. Para perceber esta resposta do emissor a um ACK duplicado, devemos saber responder a porque é que o recetor envia um ACK duplicado primeiro. A Figura 9.10 sumaria a política de criação de ACK's pelo TCP (segundo as normas RFC 1122 e RFC 2581).

< RFC 1122, RFC 2581

evento	ação do recetor TCP
Chegada em ordem de um segmento com número de sequência esperado. Todos os dados até ao número de sequência já foram reconhecidos.	Reconhecimento atrasado (delayed ACK). Espera até 500ms pela chegada de outro segmento em ordem. Se o próximo segmento na mesma ordem não chegar neste intervalo, envia um ACK.
Chegada em ordem de um segmento com número de sequência esperado. Um outro segmento em ordem aguarda transmissão de um ACK.	Imediatamente envia um só ACK acumulado, reconhecendo ambos os segmentos em ordem.
Chegada fora de ordem de um segmento com número de sequência mais alto que o esperado. Intervalo detetado.	Imediatamente envia um ACK duplicado, indicando o número de sequência do byte seguinte esperado (que é o limite inferior do intervalo).
Chegada de um segmento que completa parcial ou inteiramente um intervalo nos dados recebidos.	Imediatamente envia um ACK, com o segmento a iniciar no limite inferior do intervalo.

figura 9.10

Quando um recetor recebe um segmento com um número de sequência que é maior que o seguinte, esperado e em ordem, deteta um intervalo da sequência de dados -

isto é, um segmento em falta. Este intervalo pode ser resultado de segmentos perdidos ou reordenados na rede. Sendo que o TCP não usa reconhecimentos negativos, o recetor não pode enviar um ACK negativo de volta para o emissor. Ao invés, ele simplesmente repete o reconhecimento (isto é, envia um ACK duplicado) para o último byte ordenado que foi recebido.

Porque o emissor envia frequentemente um largo número de segmentos verificados, se um segmento se perde, então é provável que hajam muitos segmentos que voltem como ACK's duplicados. Se o emissor TCP recebe **três ACK duplicados** para o mesmo segmento de dados, este toma como indicação de que o segmento já fora reconhecido 3 vezes e fora perdido. No caso em que três ACK duplicados são recebidos, o emissor TCP faz um **fast retransmit** (inglês para retransmissão rápida), segundo a norma RFC 2581, retransmitindo o segmento perdido antes que o cronómetro incluído no segmento esgote. Este cenário está representado na Figura 9.11.

três ACK duplicados

fast retransmit
[< RFC 2581](#)

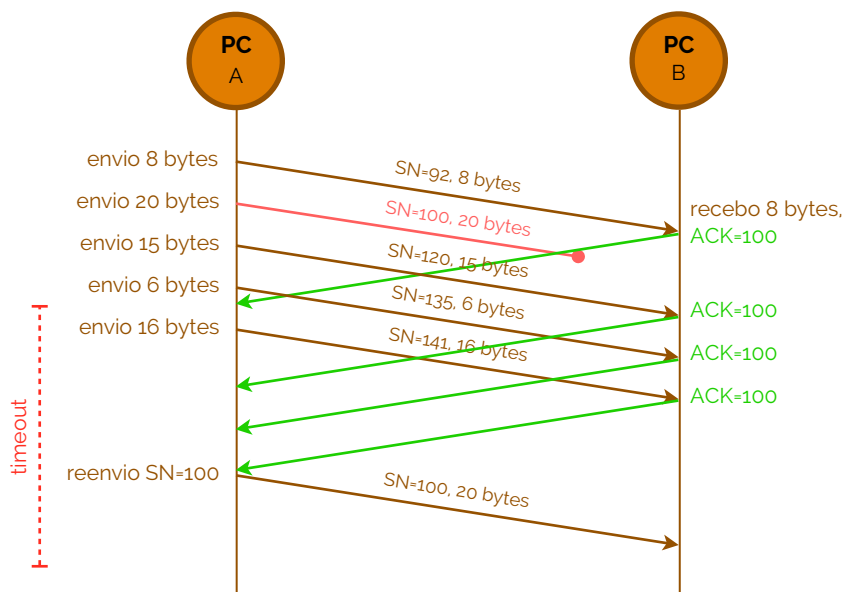


figura 9.11

Para o TCP fast retransmit, o seguinte excerto de código (a castanho) corrige o evento de reconhecimento recebido da Figura 9.6, na Figura 9.12.

```
evento: reconhecimento recebido, com campo ACK igual a y
se (y > SendBase) {
    SendBase = y
    se (existe algum segmento não reconhecido)
        iniciar cronómetro
} caso contrário {
    incrementar número de ACK duplicados recebidos por y
    se (número de ACK duplicados recebidos por y == 3)
        retransmitir segmento com número de sequência y
}
break;
```

figura 9.12

Recuperação de erros em TCP: Go-Back-N ou Selective Repeat?

Mas afinal o TCP usa Go-Back-N ou Selective Repeat como algoritmo de recuperação de erros? Relembremos que os reconhecimentos TCP são cumulativos e corretamente recebidos, contudo, segmentos fora de ordem não são reconhecidos individualmente pelo recetor. Consequentemente, como vimos na Figura 9.6 o emissor apenas precisa de manter o número de sequência mais baixo de um byte transmitido mas não reconhecido (SendBase) e o número de sequência do byte seguinte a ser enviado (NextSeqNum). Neste sentido, o TCP parece incluir-se no estilo do Go-Back-N, mas ainda há algumas diferenças entre o TCP e o Go-Back-N: considere-se o que acontece quando o emissor envia a sequência de segmentos 1, 2, ..., N e todos os segmentos chegam na ordem correta sem qualquer erro no recetor. Mais, suponhamos também que o reconhecimento

para o pacote $n < N$ perdem-se, mas os restantes reconhecimentos $N - 1$ chegam antes dos respetivos timeouts. Neste exemplo, GB-N iria retransmitir não só o pacote n , como também todos os pacotes subsequentes $n+1$, $n+2$, ..., N . O TCP, por outro lado, iria retransmitir no máximo um segmento, nomeadamente, o segmento n . Mais ainda, o TCP não iria retransmitir sequer o segmento n se o reconhecimento do segmento $n+1$ chegasse antes do timeout para o segmento n . Pode-se então dizer que o TCP possui um mecanismo de recuperação de erros híbrido entre Go-Back-N e Selective Repeat.

Controlo de fluxo em transmissões TCP

Como já foi referido anteriormente, numa transmissão TCP existem duas estruturas posicionadas, cada uma em cada extremo do canal de comunicação: um buffer de receção e um buffer para o emissor. Quando a ligação TCP recebe bytes que estão em ordem e corretos, estes são colocados no buffer. A aplicação associada irá depois ler os dados do buffer, mas não necessariamente quando os dados chegam. De facto, esta aplicação até pode estar ocupada com outra tarefa e pode nem sequer fazer uma tentativa de leitura dos dados até muito tempo depois dos dados terem sido recebidos. Se a aplicação é relativamente lenta a ler os dados, o emissor pode muito facilmente fazer um overflow no buffer na receção, enviando demasiados dados, demasiado rápido.

O TCP fornece então a capacidade de **controlo de fluxo** para as suas aplicações, de forma a eliminar a possibilidade do emissor fazer overflow ao buffer do recetor. Pode-se então dizer que o controlo de fluxo do TCP é uma forma de igualar a velocidade de escrita no buffer com a velocidade de leitura da aplicação no buffer. Como já foi denotado antes, um emissor TCP pode acionar mecanismos especiais devido a congestionamentos da rede IP - esta forma de controlo de emissão é também referida como **controlo de congestionamento**.

A capacidade de controlo de fluxo está presente quando o TCP faz com que o emissor mantenha uma variável de seu nome **janela de receção**. Informalmente, a janela de receção é usada para dar ao emissor uma ideia de quanto espaço livre existe no buffer de receção. Como o TCP é full-duplex, o emissor em cada um dos lados mantém uma janela de receção distinta. Vejamos melhor, então, como é que a janela de receção funciona: para tal consideremos uma ligação entre um computador A que envia um ficheiro de largas dimensões para um computador B através de uma ligação TCP; o computador B aloca um buffer de receção para esta ligação (denota-se o nome do tamanho deste buffer como `RcvBuffer`). Definem-se então, as seguintes variáveis com as quais iremos trabalhar, sendo que, de tempos em tempos, o processo da aplicação no computador B lê do buffer:

- `LastByteRead` - o número do último byte na sequência de dados que foi lido pelo buffer da aplicação em B;
- `LastByteRcvd` - o número do último byte na sequência de dados que chegou da rede e que foi colocado no buffer recetor em B.

Como o TCP não permite que haja overflow dos buffers alocados, devemos ter uma relação em que $\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$. A janela de receção, denotada de `RcvWindow`, está então configurada de modo a que o seu tamanho seja equivalente ao espaço livre no buffer, pelo que $\text{RcvWindow} = \text{RcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})$. Como o espaço livre no buffer varia com o tempo, `RcvWindow` é dinâmico. Esta variável encontra-se ilustrada na Figura 9.13.

Como é que a ligação usa a variável `RcvWindow` de forma a fornecer um serviço de controlo de fluxo? O computador B informa o computador A sobre a quantidade de espaço livre que tem no seu buffer, colocando o seu valor atual de `RcvWindow` no campo de janela de receção de cada segmento que envia para A (campo `WINDOW`, do segmento TCP - ver §§Transmission Control Protocol (TCP)). Inicialmente o computador B configura $\text{RcvWindow} = \text{RcvBuffer}$ - note-se que para tal acontecer, o computador B terá que manter-se à escuta de várias outras variáveis específicas e orientadas à ligação.

controlo de fluxo

**controlo de
congestionamento
janela de receção**

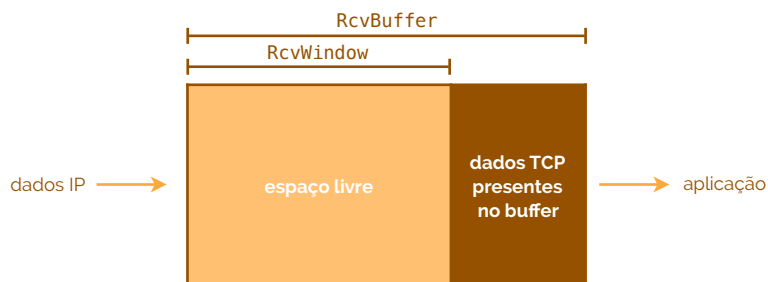


figura 9.13

Já o computador A mantém registo de duas variáveis: `LastByteSent` e `LastByteAcked`, respetivamente, significando o último byte enviado e o último byte reconhecido. Note-se que a diferença entre estas duas variáveis (`LastByteSent - LastByteAcked`) é a quantidade de dados que não foram reconhecidos que A enviou para a ligação. Mantendo esta quantidade de dados não reconhecidos menor que o valor de `RcvWindow`, A garante que não ocorre overflow do buffer recetor em B. Assim, o computador A pode também garantir, durante o decorrer da transmissão TCP, que $\text{LastByteSent} - \text{LastByteAcked} \leq \text{RcvWindow}$.

Só há um pequeno senão neste nosso esquema de estudo. Para nos apercebermos de tal acontecimento, primeiro consideremos que o buffer de receção do computador B fica cheio, logo, que a sua variável `RcvWindow = 0`. Depois de informar este valor ao computador A, suponhamos também que B não tem nada mais para enviar para A. O que poderá correr mal? Enquanto o processo da aplicação em B esvazia o buffer, o TCP não envia novos segmentos com os novos valores de `RcvWindow` para o computador A. De facto, o TCP envia um segmento para o computador A se e só se tem mais dados a enviar ou se tem um reconhecimento a cumprir. Consequentemente, A nunca é informado que já há mais espaço no buffer de receção em B - o computador A está agora bloqueado e não pode transmitir mais dados. Para resolver este problema, as especificações do TCP requerem que o computador A continue a enviar segmentos com um byte quando a janela de receção de B é zero. Estes segmentos serão reconhecidos pelo recetor. Eventualmente, o buffer irá começar a esvaziar e os reconhecimentos irão conter valores não nulos de `RcvWindow`.

Controlo de congestionamento TCP

Embora já referido, de um modo mais informal, muitas fontes de dados podem estar a transmitir, em simultâneo, demasiados dados, muito rápido para a rede em questão e esta pode não conseguir suportar tanto carregamento o quão seria esperado. Note-se que isto é diferente de controlar o fluxo da rede, pois, antes, estamos a controlar o TCP dependendo do **congestionamento** atual da rede. Algumas das consequências destes congestionamentos podem ser grandes atrasos (podem começar a formar-se filas enormes de atendimento nos buffers dos routers) ou até mesmo a perda de pacotes (caso os buffers dos routers não consigam suportar mais pacotes em fila de espera).

congestionamento

Neste controlo de congestionamento iremos ter duas abordagens possíveis: controlo nos terminais da ligação (ambos os extremos), onde não haverá propriamente feedback proveniente da rede, mas antes inferida das perdas e atrasos observados nos terminais (abordagem usada pelo TCP); controlo orientado à rede, onde são os routers que informam os sistemas terminais qual a taxa de transferência específica a que os terminais devem enviar os dados.

O primeiro caso, então, o que o TCP usa, é controlar o congestionamento através dos terminais e dos pacotes que perde ou dos atrasos dos pacotes que chegam. Esta aproximação denomina-se de **Additive-Increase, Multiplicative-Decrease (AIMD)**. Basicamente, o que se faz é aumentar a taxa de transmissão (o tamanho da janela de receção) testando a largura de banda útil, até que ocorra uma perda. Podemos então dizer que a ideia que reside por detrás de toda a lógica TCP para o controlo de congestionamento é que o emissor deve reduzir a sua taxa de transmissão (diminuindo o tamanho da sua janela de congestionamento, que aqui iremos denominar de `CongWin`)

**Additive-Increase,
Multiplicative-Decrease,
AIMD**

quando uma perda ocorre. Sendo que outras ligações TCP que podem estar a partilhar o mesmo canal entre routers congestionados também podem passar por problemas de perdas de pacotes, estes, também estão candidatos a reduzir a sua carga de transmissão de pacotes para a rede, diminuindo os seus valores de *CongWin*. O efeito geral é que, para fontes com caminhos que passam por routers congestionados deve ser produzida uma redução das suas taxas de transmissão para a rede, o que deve aliviar o próprio congestionamento nos routers. A questão principal que se coloca agora é: mas em quanto é que o TCP deve reduzir a sua taxa de transmissão quando um evento de perda de pacotes ocorre? É aqui que entra a segunda parte do algoritmo - *multiplicative-decrease* -, isto é, reduz por metade cada vez que isto acontece. Assim, se o valor de *CongWin* de um emissor TCP é atualmente 20 Kb e uma perda é detetada, então passará a metade, isto é, a 10 Kb. Caso outro evento ocorra, então esse valor voltará a ser cortado em metade - logo passará a 5 Kb.

Tendo descrito como é que um emissor TCP diminui a sua taxa de transferência face a congestionamento, é agora natural considerar como é que o TCP aumenta de novo a sua taxa, quando deixa de haver congestionamento, isto é, quando os reconhecimentos (ACK) chegam para dados ainda não reconhecidos. O rácio para um aumento de taxa é tal que se não for detetado qualquer congestionamento, então é provável que haja largura de banda livre (não usada) que possa ser usada pelo TCP. Em tais circunstâncias, o TCP aumenta a sua janela de congestionamento lentamente, testando cautelosamente largura de banda livre extra, na sua ligação entre terminais. O emissor TCP faz isto incrementando o valor de *CongWin* devagar, isto é, por cada vez que recebe um reconhecimento, com o objetivo de aumentar *CongWin* 1 MSS⁵ por cada tempo de ida-e-volta (RTT), pela norma RFC 2581. Isto pode ser feito de variadas formas: uma aproximação mais comum é o emissor TCP aumentar *CongWin* por $MSS * (MSS / CongWin)$ bytes, sempre que um novo reconhecimento é cumprido. Por exemplo, se o MSS for igual a 1460 bytes e a janela de congestionamento tiver 14 600 bytes, e 10 segmentos estiverem a ser enviados nos RTT estimados, por cada reconhecimento que chegue (assumindo que há apenas um ACK por segmento) a janela aumenta por 1/10 MSS, e conseqüentemente, depois de todos os reconhecimentos dados, o valor da janela de congestionamento terá aumentado pelo MSS, conforme se deseja.

Em suma, um emissor TCP aumenta aditivamente (*additively increases*) a sua taxa quando percebe que a ligação está livre de congestionamentos e diminui multiplicativamente (*multiplicatively decreases*) quando deteta (através de um evento de perda) que o caminho está congestionado. Por esta razão, o controlo de congestionamento TCP é muitas vezes referido pelo algoritmo AIMD, de *Additive-Increase, Multiplicative-Decrease*, como já tínhamos visto. A fase linear de aumento do controlo de congestionamento TCP é conhecido como **congestion avoidance**. O valor de *CongWin* entra então em ciclos no qual aumenta linearmente e de repente desce para metade do seu valor, formando um gráfico em forma de serrote em conexões ativas de TCP, como podemos ver na Figura 9.14.

congestion avoidance

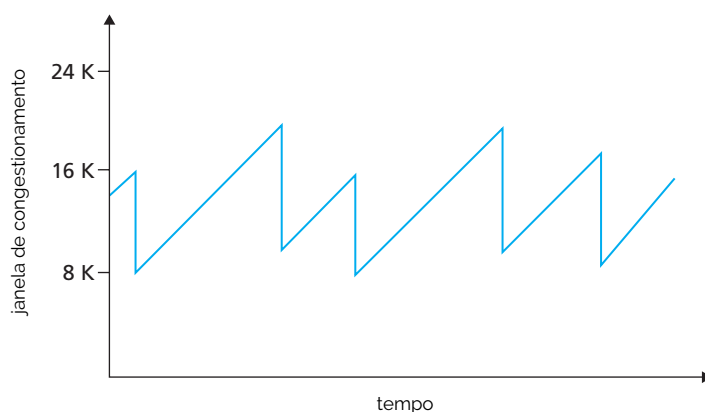


figura 9.14

⁵ MSS significa tamanho máximo de segmento (*maximum segment size*).

Quando uma ligação TCP é iniciada, o valor de `CongWin` é tipicamente inicializado a 1 MSS, pela norma RFC 3390, resultando numa taxa de transmissão inicial de aproximadamente MSS/RTT . Como exemplo, se o $\text{MSS} = 500$ bytes e $\text{RTT} = 200$ ms, então a taxa seria inicializada a 20 kbps. Como a largura de banda disponível, neste ponto, é deve ser bem maior que o valor de MSS/RTT , seria um desperdício de tempo e recursos livres aumentar linearmente a taxa de transmissão, pelo que se passou a aumentar **exponencialmente**, duplicando o seu valor de `CongWin` por cada RTT. Assim, o emissor TCP continua a aumentar a sua taxa de emissão exponencialmente até que há um evento de perda, o que despoletará um corte em metade, da janela de congestionamento e, só aqui, é que a taxa volta a crescer linearmente. Assim, nesta fase inicial, denominada de **slow start**, o emissor TCP começa por transmitir a uma taxa muito baixa (daí o termo *slow start*), mas aumenta-a exponencialmente no tempo (Figura 9.15). O emissor gera o crescimento exponencial aumentando o valor de `CongWin` 1 MSS cada vez que um reconhecimento é cumprido.

← RFC 3390

exponencialmente

slow start

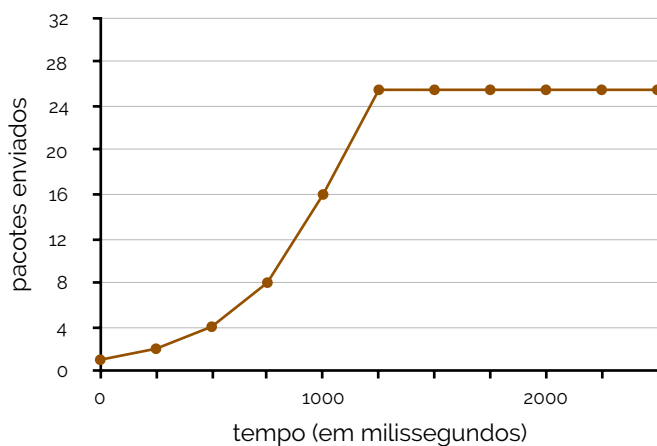


figura 9.15

Como podemos ver também na Figura 9.16, o TCP envia o primeiro segmento e aguarda pelo seu reconhecimento. Se este segmento for reconhecido antes de um evento de perda, então o TCP aumenta a janela de congestionamento por 1 MSS e envia dois segmentos. Se estes segmentos forem ambos reconhecidos antes de algum evento de perda, o emissor aumentará, de novo, a janela de congestionamento por 1 MSS para cada um dos segmentos reconhecidos e enviará 4. Este procedimento continuará até que atinja o seu limite e se mantenha num valor constante (caso não aconteçam eventos de perda) ou volte a metade, caso aconteça um evento de perda.

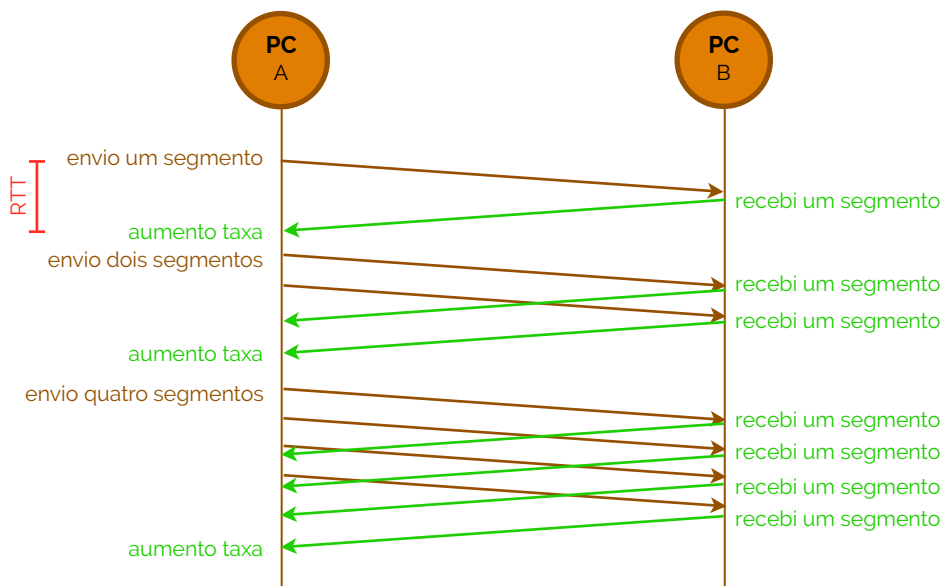


figura 9.16

Há pouco referimos que a janela de congestionamento inicia um slow start até um evento de perda, sobre o qual é cortada a meio, onde o gráfico de serrote se começa a formar, devido ao AIMD. Apesar de ser verdade, a estória não está contada até ao fim, ainda, porque na verdade, depois de três ACK duplicados o TCP comporta-se da forma que vimos, mas depois de um evento de timeout, o emissor entra em modo de slow start de novo, isto é, coloca o valor de MSS a 1 faz este crescer exponencialmente. A janela continua em crescimento exponencial até que *CongWin* atinja metade do valor que teve antes do timeout. Neste ponto, *CongWin* cresce linearmente, tal como se fosse resposta a três ACK duplicados.

Para que isto possa acontecer, o TCP mantém uma variável denominada de *Threshold* que determina o tamanho da janela com o qual o slow start deve terminar da próxima vez que seja executado e se ligue o *congestion avoidance*. Esta variável, inicialmente está configurada com um valor muito alto, pelo que não faz efeito no arranque. Quando há um evento de perda, esta variável toma imediatamente metade do valor de *CongWin*. Por exemplo, se a janela de congestionamento for de 20 Kb antes de um evento de perda, então o valor de *Threshold* passa para 10 Kb e manter-se-á até ao próximo evento de perda. Na Figura 9.17 podemos ver então uma tabela onde se descreve, muito sumariamente, qual o comportamento das variáveis *CongWin* e *Threshold* por estado e evento.

estado	evento	controlo de cong. TCP no emissor	notas
slow start (SS)	reconhecimento recebido de dado não reconhecido	$\text{CongWin} = \text{CongWin} + \text{MSS}$ se ($\text{CongWin} > \text{Threshold}$) passar estado = "CA"	resulta na duplicação de <i>CongWin</i> por cada RTT
congestion avoidance (CA)	reconhecimento recebido de dado não reconhecido	$\text{CongWin} = \text{CongWin} + \text{MSS} * (\text{MSS} / \text{CongWin})$	additive increase, resulta do aumento de <i>CongWin</i> por 1 MSS em cada RTT
SS ou CA	evento de perda detetada por três ACK duplicados	$\text{Threshold} = \text{CongWin} / 2$ $\text{CongWin} = \text{Threshold}$ passar estado = "CA"	fast recovery, implementando multiplicative decrease. <i>CongWin</i> não diminuirá menos que 1 MSS.
SS ou CA	evento de timeout	$\text{Threshold} = \text{CongWin} / 2$ $\text{CongWin} = 1 \text{ MSS}$ passar estado = "SS"	entrar em slow start
SS ou CA	evento de ACK duplicado	aumentar contagem de ACK duplicados para o segmento reconhecido	variáveis <i>CongWin</i> e <i>Threshold</i> ficam intactas

figura 9.17

Neste ponto já deve ser normal entender como o TCP se comporta de formas diferentes para eventos de timeout ou de três ACK duplicados. Mas mais especificamente, porque é que um emissor TCP se comporta de forma tão conservadora depois de um evento de timeout, reduzindo a sua janela para 1 MSS, se quando recebe um triplo ACK duplicado só corta a janela em metade? É interessante verificar que uma versão anterior do TCP, conhecida pelo **TCP Tahoe**, incondicionalmente corta a janela de congestionamento para 1 MSS e entra em slow start, depois de qualquer evento de perda. A versão mais recente, a **TCP Reno**, cancela o slow start depois de um triplo ACK duplicado. A filosofia por detrás de tudo isto está no facto de, mesmo que um pacote tenha sido perdido, a chegada de três ACK duplicados significa que, alguns segmentos foram recebidos no emissor. Assim, diferentemente do caso de timeout, a rede mostra-se capaz de entregar alguns segmentos, mesmo que outros estejam a ser perdidos no congestionamento. Este cancelamento do slow start depois de um triplo ACK duplicado é denominado de **fast recovery** (recuperação rápida).

A Figura 9.18 mostra então a evolução do TCP em termos da janela de congestionamento, para tanto a versão Reno como para a versão Tahoe. Nesta figura inicialmente o *Threshold* é 8 MSS. Para as primeiras 8 rondas de transmissão, Tahoe e Reno têm ações idênticas. A janela de congestionamento sobe exponencialmente durante o slow start e atinge o valor de *Threshold* na quarta ronda de transmissão. A janela de

TCP Tahoe**TCP Reno****fast recovery**

congestionamento depois cresce linearmente até um triplo ACK duplicado, mesmo antes da oitava ronda de transmissão. Note-se que a janela de congestionamento é $12 * MSS$, quando este evento de perda ocorre. O *threshold* é então modificado para $0.5 * CongWin = 6 * MSS$. Pelo TCP Reno, a janela de congestionamento é dada por $CongWin = 6 * MSS$ e depois cresce linearmente - já pela parte do TCP Tahoe, a janela de congestionamento volta a 1 MSS e cresce exponencialmente segundo o slow start. Note-se que grande parte das implementações TCP usam o algoritmo do TCP Reno.

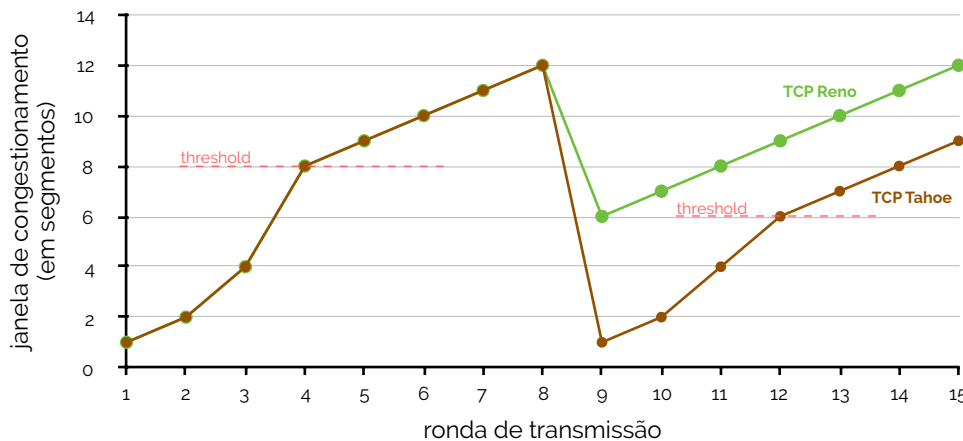


figura 9.18
TCP Tahoe vs. TCP Reno

10. Camada de Aplicação

As aplicações de rede são as razões da criação da área de estudo de redes computacionais - se não pudéssemos conceber nenhuma aplicação útil, então não haveria a necessidade de estudar esta área, nem de ela própria existir. Tendo já os conhecimentos das camadas de dados, rede e transporte, agora estamos prontos para avançar e discutir pormenores numa camada acima - a **camada de aplicação**. Primeiramente iremos definir aspetos-chave desta camada, tais como os serviços requeridos por determinadas aplicações, clientes e servidores, processos, ...

camada de aplicação

Aspetos-chave sobre aplicações

Suponhamos que temos uma ideia para uma aplicação que necessite de usar uma rede. No centro do desenvolvimento da aplicação está a escrita de programas que executem em diferentes sistemas terminais mas em comunicação um com o outro. Por exemplo, quando usamos a Internet geralmente há dois programas distintos que correm um com o outro: o browser que corre no computador do utilizador que acede à Web e o programa de servidor Web, que corre remotamente, num outro computador servidor.

As comunicações de uma rede têm de se reger a determinadas **arquiteturas**. Aqui iremos discutir quais são as arquiteturas dentro da camada de aplicação, mas nas camadas anteriores as suas arquiteturas serão discutidas na disciplina de Arquitetura de Redes (a3s2) e, mais à frente e mais aprofundadamente em termos globais de camadas, em Arquiteturas de Redes Avançadas (a4s1). Seguindo, em termos de aplicações podemos ter uma de duas arquiteturas: arquitetura cliente-servidor ou a arquitetura peer-to-peer (abreviadamente P2P).

arquiteturas

Numa arquitetura **cliente-servidor** há sempre um terminal que está sempre ligado, denominado de **servidor**, e que fornece um vasto conjunto de serviços dependendo da aplicação em causa a **clientes** (outros terminais que possam aceder ao servidor). Um exemplo clássico é uma aplicação Web que esteja sempre ligada, como um servidor da Google que está sempre ligado, pelo que seja a que tempo for, podemos fazer uma pesquisa na sua página Web que obtemos sempre resposta, dado que o servidor (neste caso é um conjunto de servidores, denominado de **server farm**) está sempre ativo. Por estar sempre ativo, e também sendo uma característica das arquiteturas cliente-servidor, o servidor tem

cliente-servidor
servidor
clientes

server farm

um endereço IP atribuído que é estático e bem-conhecido pelos outros terminais. Alguns exemplos que usam arquiteturas servidor-cliente são a Web, FTP, Telnet ou até mesmo o e-mail.

Por outro lado, temos as arquiteturas **peer-to-peer** (ou **P2P**). Nestas arquiteturas há uma probabilidade mínima ou inexistente em equipamentos que estejam sempre ativos e que sirvam de servidores. Pelo contrário, uma relação peer-to-peer é estabelecida entre pares de terminais intermitentes, denominados de **peers**. Os peers não são património de nenhum fornecedor de serviço, antes, são computadores que são controlados por utilizadores, sendo que a maior parte dos serviços que funcionam com base em P2P estão instituídos em universidades, casas e escritórios. Estas aplicações que usam este tipo de arquitetura geralmente possuem uma forte vertente de distribuição de ficheiros, como o BitTorrent, ou telefonia, como o Skype, ou até mesmo IPTV, como o Kodi.

Embora tenhamos referido duas arquiteturas distintas é importante frisar que é possível instituir arquiteturas híbridas, isto é, produto da combinação entre ambas as arquiteturas que estudámos.

Como já referimos antes, as aplicações, quando são executadas, em cada terminal são identificadas com um ou mais números, denominados de números de processo. Estes **processos** são identificadores de uma aplicação em execução. A importância deste identificador nesta área provém pelo facto de fazer parte da integração e partilha de informação entre as quarta e quinta camadas (transporte e aplicação, respetivamente). Como já tivemos oportunidade de verificar antes, por exemplo, numa ligação TCP, as aplicações leem os dados contidos nos buffers de receção, tal como também escrevem nos buffers de emissão - esta leitura escrita é garantida por uma ligação que faz corresponder um processo (a aplicação) a um transporte a que lhe damos o nome de **socket**. Se continuarmos com a analogia das docas que antes referimos, se considerarmos o TCP ou UDP como um navio de carga, o processo é o identificador do porto de carga, onde uma pessoa (um socket) está à espera do navio para descarregar o seu conteúdo.

Agora estamos em perfeitas condições para designar quais os melhores protocolos de transporte (considerando apenas UDP e TCP) para as seguintes aplicações: e-mail, acesso remoto a terminal, Web, transferência de ficheiro, streaming de multimédia e telefonia (videochamada, por exemplo). Na Figura 10.1 temos uma tabela com essa informação, relembrando que o TCP recupera segmentos perdidos, enquanto que o UDP segue ignorando pacotes perdidos.

aplicação	protocolo de transporte	exemplo de aplicação
e-mail	TCP	SMTP
acesso remoto a terminal	TCP	Telnet
Web	TCP	HTTP
transferência de ficheiros	TCP	FTP
streaming multimédia	TCP ou UDP	HTTP, RTP
telefonia	tipicamente UDP	Skype, RTP

figura 10.1

File Transfer Protocol (FTP)

Numa sessão típica FTP o utilizador encontra-se num terminal e quer transferir ficheiros para outro terminal remoto. De forma a poder aceder à conta remota, o utilizador deve fornecer uma identificação através de um nome de utilizador e provavelmente uma palavra-passe. Depois de processada a autenticação, o utilizador poderá efetuar transferências de ficheiros locais para o computador remoto e vice-versa. Como se pode ver na Figura 10.2, o utilizador interage com o FTP através de um agente FTP. O utilizador primeiro tem de informar a que host é que ele pretende aceder, fazendo com que a face cliente FTP acione uma ligação TCP com o servidor FTP, no terminal remoto. O utilizador então depois fornece as credenciais de autenticação e estes são enviados por FTP como comandos. Uma vez que o servidor tenha autorizado o utilizador a aceder, este copia um ou mais ficheiros locais no terminal remoto (ou vice-versa).

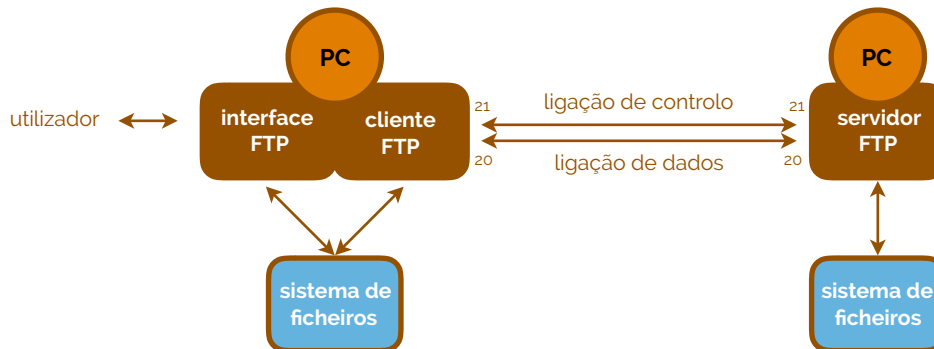
O **FTP** é então um protocolo de transferência de ficheiros que corre sobre o TCP. **FTP**

Este protocolo interage sobre duas máquinas, através de duas ligações TCP: uma para dados e outra para controlo. Numa ligação FTP normal (considerada como **ativa**), a ligação TCP de dados é efetuada entre os portos 20 de cada máquina, enquanto que a ligação TCP de controlo é feita entre os portos 21 de cada máquina. Dada esta característica de co-existirem duas ligações TCP separadas, diz-se que o FTP envia a sua informação de controlo **fora-de-banda** (em inglês, *out-of-band*).

ativa

fora-de-banda

figura 10.2



Quando um utilizador inicia uma ligação FTP com um servidor remoto, o lado cliente do FTP (o utilizador) primeiro inicia a ligação de controlo TCP com o lado servidor (computador remoto) na porta 21. O lado cliente do FTP envia a identificação do utilizador e a sua password pelo canal de controlo. Este lado, também envia, pela ligação de controlo, comandos para alterar o diretório remoto. Quando o servidor recebe um comando para uma transferência de ficheiro pelo canal de controlo (do ou para o computador remoto), este lado inicia uma ligação TCP de dados para o lado cliente. O FTP envia então exatamente um ficheiro pela ligação de dados e fecha o canal. Se, durante a mesma sessão, o utilizador quiser transferir outro ficheiro, então o FTP abrirá uma ligação para ele. Assim, com o FTP, a ligação de controlo permanece aberta durante todo o decorrer da sessão do utilizador, enquanto estiver aberta também, mas uma ligação de dados é criada para cada ficheiro que é transferido numa sessão, isto é, as ligações de dados são **não-persistentes**.

não-persistentes
estado

Durante uma sessão, o servidor FTP deve manter o seu **estado** acerca do utilizador, isto é, o servidor FTP deve associar a ligação de controlo com uma conta específica de utilizador e também deve tomar conta de qual é o diretório atual do utilizador, enquanto este navega e descobre o diretório remoto. Seguir todas estas informações acerca do estado para cada sessão que ocorre limita significativamente o número total de sessões que o FTP pode manter em simultâneo.

Alguns dos comandos que o FTP permite ao utilizador executar são os seguintes:

- **USER username** - usado para enviar o nome de utilizador do utilizador para o servidor;
- **PASS password** - usado para enviar a palavra-passe de utilizador do utilizador para o servidor;
- **LIST** - usado para pedir ao servidor que envie uma lista de todos os ficheiros que se encontram no presente diretório remoto, sendo esta enviada sob uma nova e não-persistente ligação de dados TCP, ao invés da ligação de controlo;
- **RETR filename** - usado para obter um ficheiro do presente diretório remoto, fazendo com que o terminal remoto inicie uma ligação de dados e envie o ficheiro por ela;
- **STOR filename** - usado para gravar um ficheiro no presente diretório remoto.

Por norma, há sempre uma relação unívoca entre o comando que o utilizador escreve e o comando FTP que é enviado sob a ligação de controlo TCP. Cada comando é seguido de uma resposta, enviada do servidor para o cliente. As respostas contêm um número de 3 dígitos que identifica a mensagem de estado e a descrição do estado, como:

- › 331 Username OK, password required;
- › 125 Data connection already open; transfer starting;
- › 425 Can't open data connection;
- › 452 Error writing file.

Um outro comando importante do FTP é o comando **PORT**, que especifica o endereço IP e o número do porto para o qual o outro parceiro (*peer*) deve estabelecer a ligação TCP para a transmissão de dados. Este comando tem a seguinte forma, representada na Figura 10.3.

`PORT <byte1-ip>,<byte2-ip>,<byte3-ip>,<byte4-ip>,<MSB-port-number>,<LSB-port-number>`

PORT

figura 10.3
comando PORT

Um exemplo de comando **PORT** pode ser o seguinte, para criar uma ligação de dados para o IP 192.168.8.227 com o porto 1253: `PORT 192,168,8,227,4,229`. Para obter o número do porto multiplica-se o `<MSB-port-number>` por 256 e soma-se o `<LSB-port-number>`, como neste caso, em que o número do porto é $4 * 256 + 229 = 1253$.

A utilização do comando **PORT** pode servir para criar uma ligação FTP **passiva**. Por ligação FTP passiva entende-se que, à escolha do cliente, o servidor pode abrir a ligação de dados num porto aleatório não designado a FTP ou outra aplicação. Para tal o cliente só tem de enviar o comando **PASV**, de passivo, para informar o servidor que este quer iniciar uma ligação passiva de dados com ele, ao que o servidor deve reponder com um comando **PORT** incluindo o seu endereço IP e o porto que acaba de designar para dados no servidor. Isto costuma-se usar quando se tem firewalls entre as ligações, de modo a que os dados possam contornar estes obstáculos.

passiva

PASV

Para ver uma listagem de todos os comandos disponíveis FTP, na consola FTP podemos executar o comando **HELP**.

Trivial File Transfer Protocol (TFTP)

Entre os protocolos de transporte UDP e TCP, se tivermos de escolher um para efetuar uma transferência de ficheiros, como já vimos antes, escolhemos o TCP, porque numa transferência de ficheiros nós queremos garantir que este chega inteiro, sem qualquer erro, de um computador para outro. Se usássemos o UDP, o que provavelmente iria acontecer seria perder alguma parte do ficheiro, no entanto, se usarmos o UDP para transferências de ficheiros de muito pequenas dimensões, talvez o UDP até consiga ser útil. Assim nasce o **TFTP** (inglês para *Trivial File Transfer Protocol*), que é uma versão de protocolo de transferência de ficheiros muito simplificada e usada para ficheiros de pequena dimensão. Através de uma interface entre cliente e servidor extremamente simples, um cliente TFTP pode apenas receber um ficheiro, caso saiba o nome e a localização do ficheiro, ou escrever um. Este serviço pode ser usado para configurar elementos de uma rede computacional. A maior parte dos routers e dos switches permitem que a sua configuração seja feita através de um servidor TFTP para receber os ficheiros de configuração, precisando apenas do endereço IP do servidor e do nome do ficheiro.

TFTP

Como de outra forma não estaríamos à espera, este serviço de transferência de ficheiros não suporta que o utilizador peça para listar os ficheiros do servidor, nem tem capacidade para autenticações.

Numa ligação TFTP, que corre sobre UDP, a mensagem inicial do cliente é enviada para o servidor TFTP e porta 69. Já as mensagens seguintes do servidor ficam provenientes de outro porto localmente selecionado pelo servidor, permanecendo assim até ao fim da transmissão. Por ser tão simples, o serviço TFTP usa o mecanismo de Stop & Wait, incluindo apenas 5 mensagens: Read Request (RRQ), Write Request (WRQ), Data, Acknowledgment (ACK) e Error (ERR). Na Figura 10.4 podemos ver o formato de cada uma destas mensagens.

Para fazer um **Read Request** o utilizador tem de cumprir o comando `RRQ: filename.`

Read Request

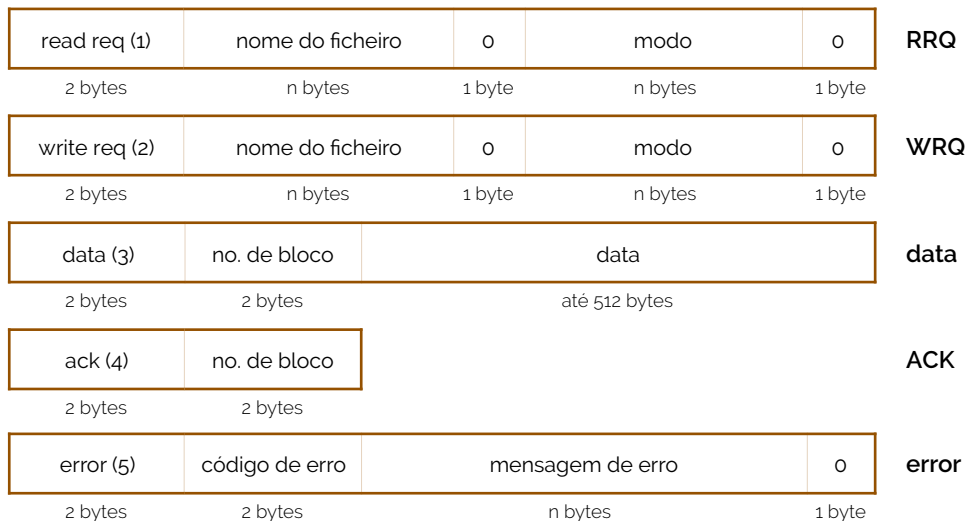


figura 10.4
mensagens TFTP

Cumprido esse comando, é levado um segmento do tipo RRQ, conforme representado na Figura 10.4 do cliente para o servidor. Diretamente, o servidor responde com o primeiro segmento de dados, que o utilizador posteriormente reconhece, depois o servidor envia o segundo segmento de dados, que o utilizador reconhece, e assim sucessivamente. Mas como é que o utilizador sabe quando é que já não há mais dados restantes? Pois bem, se repararmos bem, o segmento de dados que é enviado, conforme a Figura 10.4, permite que sejam enviados blocos com 512 bytes no máximo, por segmento. Assim, se um ficheiro tiver 1200 bytes, haverá a necessidade de enviar três segmentos: $512 + 512 + 176$. Os primeiros dois segmentos têm ambos dados com o tamanho de 512 bytes, mas o último não. Assim, o cliente saberá que se trata do fim de uma transmissão quando o último pacote não tiver dados no tamanho máximo permitido, isto é, quando o cliente receber um segmento com menos de 512 bytes de dados, então este é o último. Então e o que fazer quando se tem de enviar dados com tamanho múltiplo de 512, como 1024 bytes de dados? Quando isso acontece o TFTP tem de enviar 3 segmentos (no caso de 1024 bytes de dados), onde se somam os tamanhos dos dados da seguinte forma: $512 + 512 + 0$. Na Figura 10.5 está representada a execução do comando RRQ.

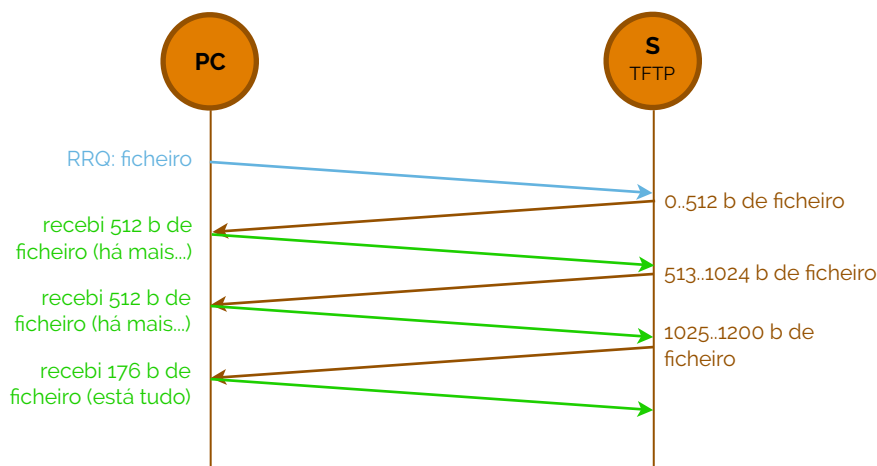


figura 10.5
RRQ

Numa sessão **Write Request** a ordem dos segmentos é diferente. Primeiro, o cliente tem de executar o comando **WRQ: filename**. Isto gera um segmento WRQ representado na Figura 10.4 que é enviado para o servidor TFTP, ao que ele responde com um reconhecimento (ACK), seguido do processamento de dados comum, como demonstra a Figura 10.6.

Write Request

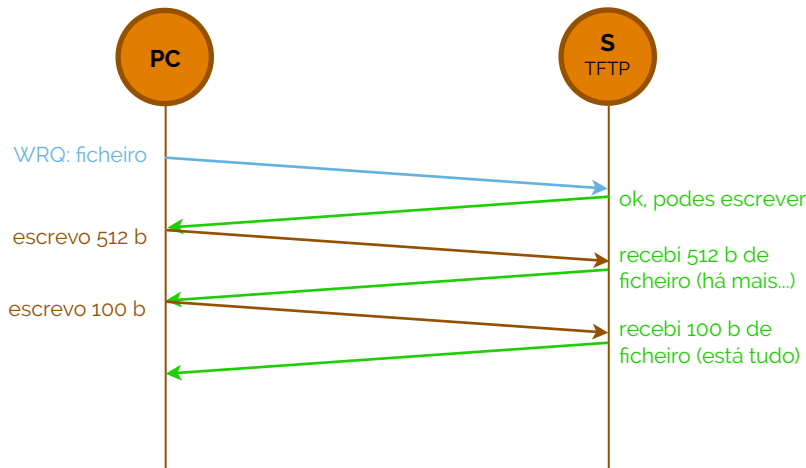


figura 10.6
WRQ

Domain Name System (DNS)

Uma das aplicações mais importantes e de que nós dependemos em muito, cada vez que acedemos à Internet é o **DNS**. Nós humanos podemos ser identificados de variadas formas: nomes, números, ... Dentro de cada uma dessas formas, ainda podemos formar outros identificadores como nome de certidão de nascimento ou nome da conta do Skype para os nomes, ou número de segurança social ou número do Cartão de Cidadão, para os números - tudo depende de um contexto, mas todas estas coisas permitem-nos identificar. Mas mesmo em termos de nomes, nós não decoramos o nome inteiro das pessoas, mas antes o primeiro e o último (quando decoramos o último) - ou então usamos alcunhas. Porquê? Porque é mais fácil decorar e é mais rápido para fazer associações entre contextos.

Tal como os humanos, os terminais na Internet também podem ter nomes (vários nomes, incluindo alcunhas) e números. Os **hostnames** (nomes dos terminais) podem ser tão variados como `google.com` ou `abc.go.com`, mas no fundo, na rede, isto não tem significado algum - não tem porque precisa de um contexto, esse que só os **servidores DNS** é que os podem fornecer. Basicamente, o DNS é um protocolo que assegura a criação e utilização de mnemónicas para fazer correspondência entre uma string e um endereço IP. Imaginemos agora se nós não tivéssemos DNS e queríamos aceder ao Google Search: teríamos de abrir o browser e saber o endereço IP 84.91.171.98, por exemplo, de cor ou teríamos uma lista para consultar. O facto de haverem servidores DNS, permite-nos que consigamos saber, através de `google.com`, qual o endereço IP que lhe está associado (ou quais).

Então, a aplicação DNS, sigla inglesa para Domain Name Server, é uma base de dados distribuída que implementa uma hierarquia de servidores DNS, assistentes e muitas vezes empregados de outros protocolos de aplicações como o HTTP, SMTP ou FTP para poder desvendar correspondências entre strings e endereços IP.

Consideremos então que estamos no nosso browser e que queremos aceder à página `www.stanford.edu`. Para tal, o browser precisará de um endereço IP para chegar à página da Universidade de Stanford, nos Estados Unidos da América. Como é que tudo isto se processa? Vejamos:

- ▶ primeiro a máquina que executa o browser ativa o cliente DNS (ele próprio);
- ▶ segundo, o browser irá extrair o hostname `www.stanford.edu` do URL⁶ e passar o hostname para o cliente DNS;
- ▶ terceiro, o cliente DNS envia uma consulta (em inglês *query*) contendo o hostname para um servidor DNS;

⁶ URL significa Uniform (ou Universal) Resource Locator e tem a forma `www.hostname`, onde `hostname` é `aaa.bbb.ccc.ddd...`

- quarto, o cliente DNS, eventualmente, recebe uma resposta, que inclui o endereço IP para o hostname (52.33.243.227);
- quinto, e por fim, uma vez o browser sabendo o endereço IP do DNS, pode iniciar a ligação TCP via HTTP no endereço designado.

Vejamos então como é que está organizada a hierarquia de servidores DNS (Figura 10.7). Num primeiro nível encontramos o **servidor raiz DNS**. Por sua vez, este servidor está ligado a um conjunto de servidores que são denominados de **TLD's** (inglês para *Top Level Domains*). Nos TLD's incluem-se todos os domínios mais importantes (mais abrangentes e usados), entre os quais temos os .com, os .org ou os .edu. Dentro de cada um destes servidores temos ainda servidores com **autoridade** (*authoritative servers*), onde podemos encontrar, por exemplo, abaixo de .edu, o stanford.edu.

servidor raiz DNS

TLD

autoridade

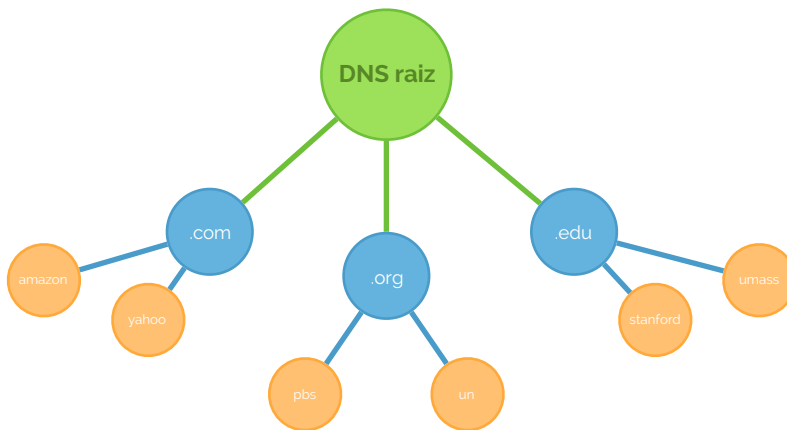


figura 10.7

Quando o DNS precisa de pesquisar um hostname, o que ele faz é invertê-lo (por exemplo o hostname `images.google.com` fica `com.google.images`) e acede ao hostname invertido concatenado com `.in-addr.arpa` e obtém um endereço. Para tal, a pesquisa é feita, por passos, lendo primeiro a primeira string antes de um ponto, cumprindo os caminhos da hierarquia - na Figura 10.7, para o endereço `un.org`, primeiro acedia-se ao TLD `.org` e só depois a `un`.

Mas nem sempre é útil, pelo menos para as instituições governamentais ou fornecedores de serviço Internet (ISP), tornar acessíveis todas as correspondências DNS no mundo. Para tal, em países como Portugal, em que há um conjunto de páginas Web que se encontram bloqueadas pelos ISP, por lei, aplica-se um **local name server**, isto é, um servidor que não tem de pertencer à hierarquia DNS, mas que faz parte do ISP e que fornece algumas correspondências pedidas de um modo diferente que faria um servidor DNS de níveis mais superiores. Quando o servidor DNS local não tiver a entrada nas suas tabelas, terá de consultar o primeiro acessível em termos hierárquicos.

local name server

Vejamos agora como é que os pedidos por correspondência de endereços passam na hierarquia. Consideremos a Figura 10.8, onde o pedido é feito de modo **iterativo**.

iterativo

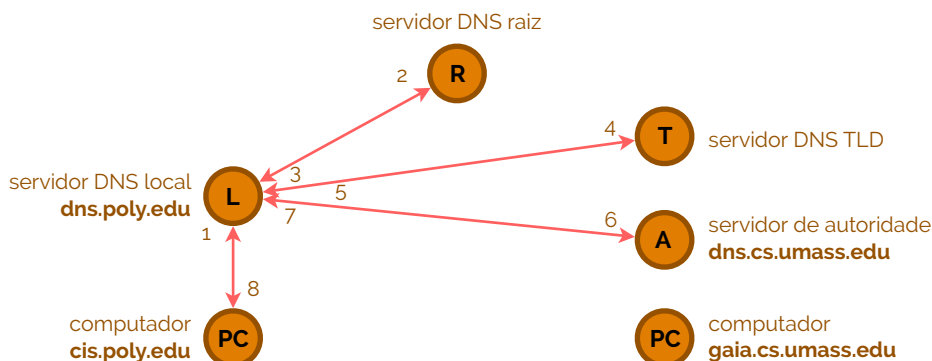


figura 10.8

Considerando a Figura 10.8, podemos verificar que os servidores são contactados a partir de um servidor DNS local. Basicamente, o que aqui acontece é que se está a perguntar, a partir do servidor local, a cada um dos outros, se conhece um determinado nome, ao que eles, caso a resposta seja negativa, dizem: "não sei quem tem esse nome, mas pergunta antes a este servidor". Esta forma de resolver o nome por DNS, apesar de funcionar, é bastante lenta e ocupa o servidor local, mantendo-o à espera de um pedido DNS. Mais, a distância que se encontra o servidor local do servidor de autoridade da figura, por exemplo, pode ser muito grande, o que irá manter o local muito tempo à espera ou mesmo sem resposta.

Uma forma de resolver os problemas das iterações é implementar um algoritmo **recursivo** (Figura 10.9). Implementando um algoritmo recursivo, na prática, o que terá de fazer é: conduzir o pedido até à raiz e descer pelo menor caminho. Embora use recursos de vários servidores, o tempo despendido para a resolução do nome DNS é muito curto.

recursivo

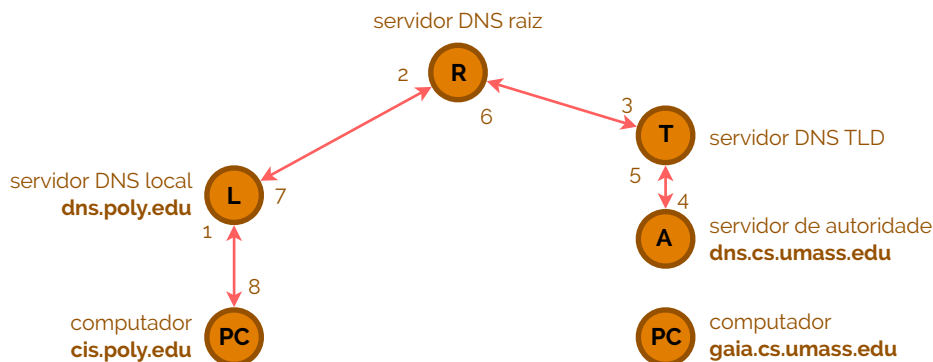


figura 10.9

Em suma, a resolução de forma recursiva é muito mais eficiente que a forma iterativa, pois minimiza o tempo entre os pedidos e as respostas, no entanto requer mais poder de processamento por parte dos servidores DNS, pelo que cada servidor terá de ser capaz de processar mais pedidos em simultâneo. Já a resolução de nomes por iterações é menos eficiente porque o tempo entre o pedido e a resposta pode ser muito largo (que o é, em média), no entanto, minimiza-se o poder de processamento requerido nos servidores DNS, pelo que cada servidor responde imediatamente a cada pedido.

Se uma determinada máquina já efetuou um pedido DNS este ficará preservado nela durante algum tempo em **cache** (depois desaparece). Assim, se de seguida, com a entrada ainda em cache, precisarmos de visitar uma página onde estivemos há pouco, não haverá pesquisa em servidores DNS pela resolução do nome. Antes, será revisitada, a tabela DNS da cache, presente localmente, na máquina.

cache

Os **registos DNS** têm diferentes tipos com propósitos distintos, sendo todos guardados sob a forma de **RR** (Resource Records) (formato (name, value, type, ttl)). Eis alguns deles: o registo do **tipo A** tem um name correspondente ao hostname e um value correspondente ao endereço IP relacionado (por exemplo, (abcde.com,84.91.171.98,A)); o **tipo AAAA** é igual ao tipo A mas para endereços IPv6; o **tipo NS** tem um name correspondente ao domínio (como foo.com) e value correspondente ao hostname de um servidor de autoridade para este domínio (por exemplo (foo.com,dns.foo.com,NS)); o **tipo MX** tem um value que corresponde ao nome do servidor de e-mail associado com o name (por exemplo, (foo.com,mail.bar.foo.com,MX)); o **tipo CNAME** tem um name que contém um alias para alguns nomes canónicos (reais) - value - (por exemplo (foo.com,relay1.bar.foo.com,CNAME)).

registos DNS

tipo A

tipo AAAA, tipo NS

tipo MX

tipo CNAME

O **protocolo DNS** possui duas mensagens (query e reply), ambas com o mesmo formato de mensagem, conforme representado na Figura 10.10. A semântica dos vários campos é a seguinte:

protocolo DNS

- os primeiros 12 bytes são o **cabeçalho DNS** que tem 6 de campos. O primeiro campo é um número de 16 bits que identifica o query. Este identificador é

cabeçalho DNS

copiado para o reply, permitindo ligar um reply ao o respetivo query. Há um conjunto de flags no campo flags, entre os quais fazem parte: um bit que indica se é reply (1) ou query (0); um bit para identificar se a mensagem reply quando provém de um DNS de autoridade; um bit que identifica se o cliente pretende usar recursividade; um bit que identifica se o servidor DNS suporta recursividade. Também existem quatro outros campos que indicam o número de ocorrências dos quatro tipos de dados que se seguem;

- a secção de questões contém informação acerca do query que está ou foi feito;
- a secção de respostas contém informação acerca das respostas a uma query previamente feito;
- a secção de autoridade possui registos de outros servidores de autoridade;
- a secção de informação adicional contém outros registos que possam ser úteis.

IDENTIFICAÇÃO	FLAGS
NÚMERO DE QUESTÕES	NÚMERO DE RESPOSTAS
NO. DE SERVIDORES DE AUTORIDADE	NÚMERO DE REGISTOS ADICIONAIS
QUESTÕES	
RESPOSTAS	
AUTORIDADE	
INFORMAÇÃO ADICIONAL	

12 bytes

figura 10.10

Consideremos agora que pretendíamos inserir um novo registo DNS na base de dados. Como é que o podemos fazer? Para tal, vamos assumir que pretendemos adicionar o endereço `networkutopia.com`. Isto pode ser feito através de um **registrar**⁷, uma entidade comercial que verifica a unicidade dos nomes de domínio, introduzindo o domínio novo nas bases de dados DNS e cobrando uma taxa pelo serviço.

registrar

HyperText Transfer Protocol (HTTP)

O protocolo-base aplicacional da Web é o **HTTP** (sigla de HyperText Transfer Protocol). Definido na norma RFC 1945 e RFC 2616, o HTTP está implementado em dois programas: um programa cliente e um servidor, ambos em execução em máquinas distintas e remotas uma da outra, trocando mensagens HTTP entre elas. O HTTP define a estrutura dessas mensagens e como é que o cliente e o servidor efetuam as trocas de mensagens.

HTTP

< RFC 1945, RFC 2616

Primeiro relembremos alguns conceitos acerca da Web que foram lecionados na disciplina de Laboratórios de Informática (a1). Uma **página Web** é um ficheiro que consiste num conjunto de **objetos**. Um objeto é simplesmente um ficheiro como uma imagem, um vídeo, um texto, áudio, ... A maior parte das páginas Web consistem de um ficheiro-base escrito numa linguagem denominada **HTML** e contém vários elementos (objetos) referenciados. Quando as páginas são publicadas, regem-se por um **URL** que tem dois componentes: o hostname do servidor que guarda o objeto e o caminho do objeto (representado a azul e a verde, respetivamente, na Figura 10.11).

página Web

objetos

HTML

URL

`http://www.someSchool.com/someDepartment/picture.gif`

figura 10.11

⁷ Uma lista completa de registrars está disponível em www.internic.net

O HTTP define como é que os clientes da Web pedem as páginas Web aos **servidores Web** e como é que estes fornecem as páginas aos clientes. Para isto acontecer, então, quando um utilizador pede uma página Web o browser envia um ou mais **HTTP Requests** para os objetos na página ao servidor. O servidor recebe o pedido e responde com um ou mais **HTTP Response** que contém os objetos pedidos. Este mecanismo encontra-se ilustrado na Figura 10.12.



figura 10.12

O HTTP usa como protocolo-base de transporte o TCP. Assim, o cliente HTTP inicializa primeiro a ligação TCP com o servidor na porta 80 (porta reservada para HTTP) e depois o browser faz comunicações com este através das interfaces dos sockets (HTTP Requests e Responses). No fim, a ligação TCP é fechada. Contrariamente ao FTP, o HTTP diz-se **stateless**, isto é, não guarda qualquer informação acerca dos últimos pedidos efetuados pelos clientes.

stateless

A ligação HTTP pode ser de um de dois tipos possíveis: persistente ou não-persistente. Uma ligação **não-persistente** provoca que no máximo um objeto seja enviado por ligação TCP - esta definição é usada na versão 1.0 do HTTP. Por outro lado, uma ligação **persistente** usa a mesma ligação TCP para a transmissão de um ou múltiplos objetos, tal como está definido por omissão na versão 1.1 do HTTP. Vejamos então o que acontece, em mais pormenor, numa ligação não-persistente, quando tentamos carregar uma página Web que consiste num ficheiro-base HTML e 10 imagens JPEG (sabemos que todos estes objetos residem no mesmo servidor), supondo também que o endereço da página é `http://www.someSchool.edu/someDepartment/home.index`. Eis o que acontece:

não-persistente

persistente

- o cliente HTTP inicia uma ligação TCP para o servidor em `www.someSchool.com` na porta 80, que é a porta pré-definida para HTTP;
- o cliente HTTP envia uma mensagem HTTP Request para o servidor através do seu socket. A mensagem request inclui o caminho `/someDepartment/home.index`.
- o servidor HTTP recebe a mensagem request no seu socket e fornece o objeto `/someDepartment/home.index` da sua unidade de armazenamento, encapsulando o objeto numa mensagem HTTP response, enviando-a para o cliente;
- o servidor HTTP informa TCP para fechar a ligação, embora o TCP ainda não feche a ligação, pois primeiro precisa de assegurar que as mensagens anteriores foram todas recebidas com sucesso;
- o cliente recebe a mensagem response. A ligação TCP termina e a mensagem indica que o objeto encapsulado é um ficheiro HTML. O cliente extrai o ficheiro da mensagem response e examina-o, encontrando referência para 10 objetos JPEG;
- os primeiros cinco passos repetem-se para cada um dos objetos referenciados.

Em termos de tempo de resposta, consideramos como tempo de ida-e-volta (RTT) o tempo que demora a enviar um pequeno pacote do cliente para o servidor e voltar. Dada esta consideração, o início de ligação TCP dura um RTT, tal como o envio do HTTP Request (juntamente com alguns bytes do HTTP Response). Se a estes dois RTT's juntarmos a conclusão do HTTP Response, fazemos um total de $2 * RTT + \text{tempo_de_transmissao}$, como podemos ver pela Figura 10.13.

O facto de necessitarmos de 2 RTT's por objeto, com uma ligação HTTP não-persistente faz com que a transmissão perca muito tempo por um só objeto.

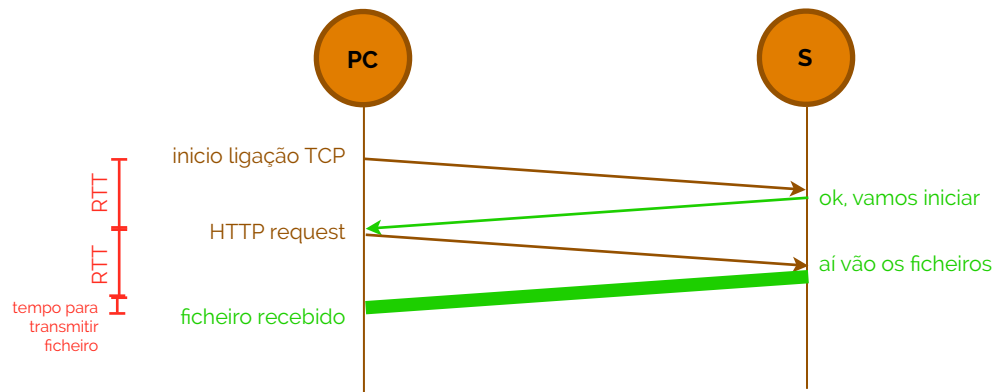


figura 10.13

Se usarmos uma ligação HTTP persistente o servidor deixará aberta a ligação TCP, permitindo que mais transmissões, de outros ficheiros, sejam feitas. No entanto não chega dizer que a ligação é persistente, porque é possível transmitir através de uma ligação desta génese com e sem **pipelining**: dizer que a ligação HTTP persistente tem pipelining (configuração por defeito do HTTP versão 1.1) é dizer que o cliente envia HTTP Requests sempre que deteta um objeto referenciado, demorando cerca de 1 RTT para todos os objetos encontrados; dizer que não possui pipelining, por sua vez, significa que o cliente só cria um novo pedido se a resposta a um pedido anterior já tenha sido recebida, necessitando, claro está, de 1 RTT por cada objeto encontrado.

pipelining

As especificações do HTTP na norma RFC 2616 (versão 1.1) incluem as definições dos **formatos de mensagens HTTP**. Existem então dois tipos de mensagens, entre as quais, mensagens de request e de reply, ambas discutidas abaixo.

formatos de mensagens

HTTP

request

Vejamos primeiro como é que se constitui a mensagem de **request**. Um exemplo desta mensagem é visível na Figura 10.14.

```
GET /someDir/page.html HTTP/1.1
Host: www.someSchool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

figura 10.14

HTTP Request

Há muita coisa que podemos aprender através de uma simples mensagem de HTTP Request. Primeiro, podemos reparar que toda a mensagem está escrita em texto sob uma codificação ASCII, de forma a que seja passível de ser lida por um humano. Segundo, podemos ver que a mensagem consiste em cinco linhas, cada uma seguida de um *carriage return* (cr) e uma *line feed* (lf) - a última linha é seguida de um cr e de uma lf adicional. Embora esta mensagem de request, em particular, tenha cinco linhas, as mensagens podem ter mais do que estas ou, pelo contrário, apenas uma linha. A primeira linha de um HTTP Request é denominada de **linha de pedido** (ou em inglês, *request line*), sendo as subsequentes denominadas de **linhas de cabeçalho** (ou em inglês, *header lines*). A linha de pedido tem três campos: o campo de **método** - podendo ter um de cinco comandos possíveis, entre eles, GET⁸, POST, HEAD, PUT e DELETE -, o campo de URL e o campo que preserva informação acerca da versão do protocolo HTTP a ser usada.

linha de pedido

linha de cabeçalho

método

Analisemos agora as linhas de cabeçalho: a linha `Host: www.someSchool.edu` especifica o terminal onde o objeto pretendido nesta mensagem reside - embora pareça desnecessário, a especificação desta linha, dado que já existe uma ligação TCP, não o é, porque a informação aqui fornecida é necessária para proxies de caches Web; a linha `Connection: closed` indica que o browser não pretende uma ligação persistente, isto é, pretende que mal a transmissão do objeto pretendido acabe, a ligação TCP se feche; o campo `User-agent` tem informações relevantes acerca do agente que enviou a mensagem HTTP, neste caso, indica

⁸ Este campo geralmente é preenchido com o comando GET (é o mais habitual). O GET serve para obter um objeto, devidamente referido no campo URL. Na Figura 10.14 mostra-se um caso em que o browser pretende o objeto /someDir/page.html.

que o utilizador está a usar uma versão 4.0 do browser Mozilla; finalmente, o campo `Accept-language` permite informar o servidor de que o utilizador pretende aceder à página que possua os conteúdos, preferencialmente, em francês (neste caso em particular, com a informação `fr`).

Tendo analisado uma mensagem HTTP Request, vejamos agora qual é a estrutura do formato de uma mensagem HTTP Request, representada na Figura 10.15.

MÉTODO		SP	URL	SP	VERSÃO		CR	LF
campo 1 do cabeçalho			SP	VALOR	CR	LF		
campo 2 do cabeçalho			SP	VALOR	CR	LF		
campo ... do cabeçalho			SP	VALOR	CR	LF		
campo n do cabeçalho			SP	VALOR	CR	LF		
CR	LF							
corpo da entidade								

figura 10.15
formato HTTP Request

Como podemos ver, no fim da mensagem HTTP Request temos o corpo da entidade. Este corpo está vazia com um método `GET`, mas é especialmente usado quando do método `POST`, usado principalmente quando o utilizador preenche algum tipo de formulário (como uma pesquisa no Google Search, por exemplo). Com este método, o utilizador ainda está a pedir uma página Web ao servidor, desta vez, dependendo da informação que partilha no campo do corpo da entidade.

Tendo visto a mensagem de HTTP Request falta-nos agora analisar a mensagem de **HTTP Response**. Esta mensagem response tem um esquema semelhante ao do exemplo particular, exposto na Figura 10.16.

HTTP Response

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 07 Jan 2016 09:56:43 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 06 Jan 2016 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
```

(data data data data data data ...)

figura 10.16
HTTP Response

Analisemos então, agora para o caso do HTTP Response, cada uma das suas linhas, começando por agrupar secções como a **linha de estado** (primeira linha que contém o estado atual da ligação HTTP, exibindo a versão a ser usada, do HTTP, e o código de estado atual - 200 significa que nada falhou, logo "OK"), seis **linhas de cabeçalho** e o **corpo da entidade** - que aqui representa o objeto requerido, por si, representado por (data data data data data data ...). Vejamos agora as linhas de cabeçalho: o servidor usa a conexão do tipo closed, isto é, não-persistente, conforme pedido pelo utilizador, há pouco; o campo `Date`: especifica a hora e a data de quando foi criado o presente HTTP Response; o campo `Server`: indica que a mensagem foi gerada por um servidor Apache (neste caso em particular); o campo `Last-Modified`: que nós veremos mais à frente com mais pormenor, indica a última vez que a página foi modificada, para que o browser não tenha de carregar todos os objetos da página através do servidor, mas antes, se já visitou antes a página, através da cache que mantém localmente; o campo `Content-Length` mantém informação acerca do tamanho do ficheiro que está a ser enviado; este, que é do tipo `Content-Type`.

linha de estado

linhas de cabeçalho
corpo da entidade

Na Figura 10.17 podemos ver, tal como vimos para o HTTP Request, qual o formato de uma mensagem HTTP Response.

VERSÃO	SP	CÓDIGO DE ESTADO		SP	FRASE		CR	LF
campo 1 do cabeçalho		SP	VALOR		CR	LF		
campo 2 do cabeçalho		SP	VALOR		CR	LF		
campo ... do cabeçalho		SP	VALOR		CR	LF		
campo n do cabeçalho		SP	VALOR		CR	LF		
CR	LF							
corpo da entidade								

figura 10.17
formato HTTP Response

Uma das partes mais interessantes do protocolo HTTP, embora seja tema para grandes discussões em termos de privacidade e segurança, é a utilização de **cookies**. Grande parte das páginas Web que usamos diariamente usam cookies, isto é, preservam informações acerca das preferências do utilizador no conjunto de ações disponíveis nas páginas Web. Então mas o HTTP não é um protocolo stateless, isto é, que não guarda qualquer informação acerca de últimas utilizações por parte do cliente? Sim, é verdade, mas os cookies não são guardados no servidor, antes, no próprio cliente e feitos acessíveis por parte dos browsers.

cookies

Os cookies são assim constituídos por quatro partes essenciais: uma linha de cabeçalho na mensagem HTTP Response; uma linha de cabeçalho na mensagem HTTP Request; um ficheiro cookie mantido na máquina cliente local, administrado pelo browser em utilização; e uma base de dados administrada pela página Web. Vejamos então, através da Figura 10.18, como é que os cookies funcionam. Suponhamos que a Ana, que acede sempre à Internet através do seu browser Safari, contacta a Amazon.com pela primeira vez. Consideremos também que antes, ela já tinha visitado a página eBay.com. Quando o request chega ao servidor da Amazon, este cria um número de identificação único e cria uma entrada na sua base de dados, indexada pelos números de identificação. O servidor então responde ao browser da Ana, incluindo no cabeçalho da mensagem HTTP `Set-cookie: 1678`. Enquanto o browser da Ana recebe a resposta HTTP, procura o campo `Set-cookie`. O browser então adiciona a linha ao ficheiro de cookie especial para o site em causa que guarda localmente. Esta linha adicionada inclui o hostname e o código de identificação.

Consideremos agora, mantendo a imagem em mente, que a Ana consulta a mesma página passado uma semana. Nesse tempo, o browser continua com a mesma informação, dado que esta se encontra preservada de forma local. Isto permite que a Amazon faça recomendações de produtos, baseando-se no histórico de páginas já visitadas anteriormente pela Ana.

Agora já percebemos porque é que os cookies são objeto de discussão em termos de privacidade: porque os cookies podem servir para identificar os utilizadores. A primeira vez que o utilizador visita a página, este fornece uma identificação de utilizador (possivelmente até mesmo o nome). Durante as ligações subsequentes, o browser passa um cabeçalho cookie para o servidor, identificando constantemente o utilizador no servidor. Basicamente, os cookies também podem servir para criar uma camada de sessão de um utilizador sobre a camada stateless do HTTP.

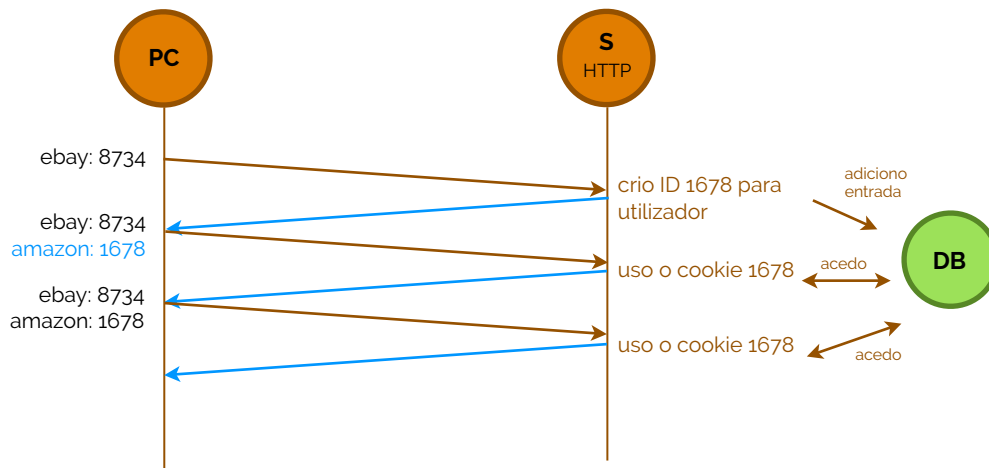


figura 10.18
cookies

Como já verificámos antes, nem sempre o browser precisa de descarregar os objetos todos a partir do servidor Web. O protocolo HTTP assegura a possibilidade de existir uma **cache Web**, pelo que tendo já carregado os objetos antes, se estes não sofreram alterações, então não há necessidade de recarregar do servidor, mas antes quase-localmente. Basicamente, o processo é o seguinte, supondo que se pretende aceder a <http://www.someSchool.edu/campus.gif>:

cache Web

- O browser cria uma ligação TCP para a cache Web e envia um pedido HTTP pelo objeto para o servidor proxy (ou servidor Web cache);
- A Web cache verifica que tem uma cópia do objeto guardada localmente: se tiver envia-a para o cliente de seguida, caso contrário este servidor abre uma ligação TCP para o servidor origem, descrito no hostname, o qual responde de volta para o Web cache server;
- A Web cache encaminha o ficheiro e grava uma cópia do mesmo (caso ainda não o tenha), terminando a ligação TCP entre si e o servidor origem.

Outra forma de reduzir o custo de ter de criar uma ligação TCP é verificar cópias dos objetos localmente, através de um **GET condicional**. Este GET funciona da seguinte forma: tendo mais um campo denominado de **If-Modified-Since**: no cabeçalho request, este verifica se os objetos pedidos foram alterados desde a data e a hora especificadas à frente, cruzando a informação com a do campo **Last-Modified** do HTTP Response. Caso se peça uma página que já foi modificada entre o tempo especificado, o browser receberá os novos objetos, caso contrário, receberá uma mensagem de estado (aviso) 304 "Not Modified".

GET condicional

Correio eletrónico (e-Mail)

O **correio eletrónico** (e-Mail) já existe desde o início da Internet, tendo sido a aplicação mais popular quando a Internet nasceu, e tem-se tornado mais e mais elaborada e potente ao longo dos anos. Tal como o correio normal, o correio eletrónico é assíncrono - isto é, as pessoas enviam e recebem mensagens quando é mais conveniente, sem ter que se coordenar horários entre várias entidades. Em contraste com o correio postal, este é mais rápido, fácil de distribuir e praticamente sem custos.

correio eletrónico

Na Figura 10.19 temos uma vista abrangente sobre o sistema de entrega de correio eletrónico da Internet. Por este diagrama podemos ver três grandes entidades: agentes, servidores de Mail e o protocolo de comunicação SMTP. Um **agente** (em inglês *user-agent*) é uma aplicação usada pelo utilizador que permite escrever, enviar, receber e ler mensagens de correio eletrónico, trocando mensagens com o **servidor Mail**. Um servidor Mail é uma aplicação que envia e recebe mensagens de correio eletrónico de/para os seus clientes. Este servidor inclui uma caixa de correio por utilizador, onde guarda as

agente

servidor Mail

mensagens destinadas a este e uma fila de espera de mensagens para outros servidores Mail que ainda não foram enviadas.

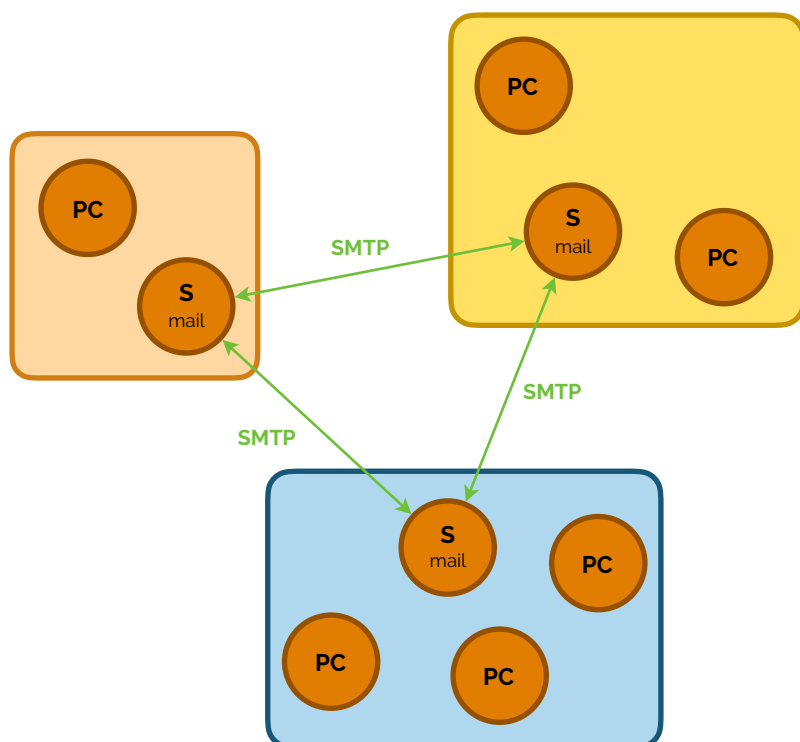


figura 10.19

O protocolo usado para o envio das mensagens de correio eletrônico é o **SMTP**, definido na norma RFC 2821. Este é o protocolo que assegura a transmissão das mensagens entre servidores Mail. O seu nome deriva de *Simple Mail Transfer Protocol* e também assegura, tal como o HTTP, a transmissão de mensagens através do agente para o servidor Mail. Por outro lado, a receção de mensagens na caixa de correio é garantida por um dos protocolos seguintes: POP3, IMAP ou HTTP.

Começemos então por analisar o SMTP. Este protocolo, como já deveria ser de esperar, corre sobre uma ligação TCP, na porta 25. As comunicações são estabelecidas pelo terminal que pretende efetuar o envio de uma mensagem (ação de **push**). Estabelecendo ligações diretas, por defeito, o servidor Mail de origem envia uma mensagem de correio eletrónico direccionada para o servidor Mail de destino, através de uma arquitetura cliente-servidor, onde o cliente envia comandos e o servidor responde aos mesmos. As mensagens trocadas são strings codificadas em ASCII e são sempre terminadas pela sequência CRLF.CRLF, onde CR é *carriage return* e LF *line feed*. Outras normas permitiram a interligação entre vários protocolos, especialmente entre o SMTP em cooperação com o **MIME** (*Multipurpose Internet Mail Extension*), definido nas normas RFC 2045 e RFC 2046, permitindo enviar mensagens de correio eletrónico com dados de diferentes tipos como imagens, vídeos, músicas ou mais...

Um dos protocolos que asseguram a receção de mensagens é, como já foi referido, o **POP3** (*Post Office Protocol* - versão 3). Este protocolo, definido na norma RFC 1939 corre sobre TCP na porta 110 e estabelece comunicações pelo terminal (agente) que pretende receber mensagens (ação **pull**). A transferência de mensagens de correio eletrónico é feita de um de dois modos: **send-and-remove**, onde a mensagem de correio eletrónico é removido depois de ser enviada pelo agente; **send-and-store**, onde a mensagem de correio eletrónico é preservada depois de ser enviada pelo agente. Este protocolo conta com três fases: **autenticação** - sobre a qual o utilizador preenche um username e uma palavra-passe; **transação** - onde o servidor Mail envia as mensagens que estão na caixa de correio, deixando o agente especificar se pretende eliminar mensagens ou não; e **atualização** - onde o servidor Mail apaga as mensagens de correio eletrónico da caixa de correio que

SMTP

◀ RFC 2821

push

MIME

◀ RFC 2045, RFC 2046

POP3

◀ RFC 1939

pull

send-and-remove

send-and-store

autenticação

transação

atualização

foram sinalizadas para tal. Este protocolo tem um forte concorrente, mais complexo e usado, de seu nome **IMAP** (*Internet Mail Access Protocol*). Em termos de diferenças para com o POP3, este corre sobre TCP no porto 143, e tem um largo conjunto de funções adicionais, entre as quais, permitir a criação de um sistema de diretórios, efetuar pesquisas, receber porções de mensagens (e não o corpo inteiro), ... Numa sessão IMAP, por outro lado, o servidor Mail pode encontrar-se em um de quatro estados: **não-autenticado** - estado inicial aquando do acesso ao servidor Mail, antes de fornecer credenciais; **autenticado** - estado seguinte após utilizador fornecer as suas credenciais; **selecionado** - o agente pode enviar qualquer comando relacionado com o paradigma das mensagens (ver, remover, transferir, ...); e **logout** - a sessão é fechada.

IMAP

não-autenticado
autenticado
selecionado
logout

11. Camada Física

Estudadas todas as quatro camadas da organização de redes computacionais que colaboram entre si para fazer com que a manipulação de dados seja cumprida da mais correta forma e sob a tutela e garantia de protocolos vários, chega agora a altura de estudar o que está por debaixo de todas estas tecnologias. Afinal de contas, temos vindo a falar em como efetuar ligações, transmitir os dados sem erros ou saber corrigir os erros, mas ainda não sabemos o que são as **ligações** e todos os paradigmas **físicos** instituídos numa rede.

ligações, físicos

Sistemas de transmissão

A **camada física** das redes computacionais é a camada responsável pela transmissão dos sinais através de um canal físico. Estas transmissões são efetuadas entre equipamentos **terminais**, que nas camadas superiores, são interpretados como origens, destinos ou meros intermediários de informação. Assim, as ligações físicas podem ser estabelecidas entre: equipamentos terminais/rede (router com router ou computador com computador) ou entre ambos (router com computador, switch com computador, ...). Existem, também, como já devemos saber da disciplina de Arquitetura de Computadores II (a2s2), dois tipos de ligação: ligações ponto-a-ponto e ligações partilhadas (também denominadas de buses).

camada física

terminais

As mensagens que temos vindo a estudar, pela camada física, são geradas através de **fontes** que podem ter duas origens: uma origem **analógica**, isto é, na qual a qualidade varia ao longo do tempo de forma contínua e natural (um exemplo é a acústica - o som); ou uma origem **digital**, isto é, uma origem na qual se fornece uma sequência de símbolos que pertencem a um número discreto de níveis/elementos. Ao longo da transmissão destes sinais, muitas vezes, há a necessidade de converter estes sinais entre analógico e digital, mas também há a necessidade de converter estes sinais entre informação e sinais elétricos (e vice-versa) - operação feita por **transducers**. Tomemos como exemplo o microfone e o auscultador - a passagem de um sinal analógico para um sinal elétrico que volta a ser convertido em analógico. Na Figura 11.1 temos dois tipos de sinais - analógico e digital - nos quais podemos ver a gama contínua de valores do sinal analógico e a gama de valores discretos do sinal digital.

fontes, analógica

digital

transducers

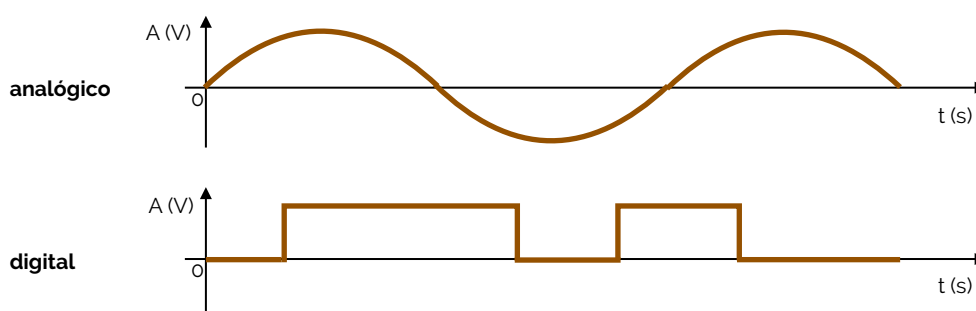


figura 11.1
sinal analógico e digital

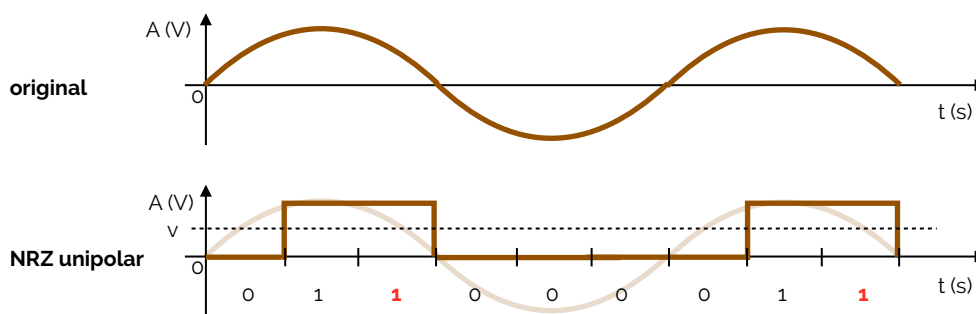
A **transmissão dos sinais** pode ser vista na Figura 11.2. Nesta figura temos um **transmissor** que adapta o sinal às características do canal de transmissão, fazendo modulação ou codificação, um **canal de transmissão** que é o meio de propagação do sinal entre o transmissor e o recetor (que pode ser de vários tipos, entre os quais cabos de cobre *twisted-pair*, cabos coaxiais, fibra ótica, espaço livre - meio ambiente) que pode contar com fatores de degradação causadores de distorções, atenuações, interferências ou ruídos. Também temos um **recetor** que processa o sinal recebido de forma a compensar os fatores de degradação a que foi sujeito no canal de transmissão, através de operações de amplificação, demodulação, descodificação ou filtragem.

Todos os sinais são **periódicos**, em que o período T é equivalente ao inverso da **frequência do sinal** (representado vulgarmente por f). Um período corresponde a uma oscilação completa do sinal. De modo a transportar sinais (que geralmente são ondas sinusoidais) as ondas são **moduladas** (modificadas) com um sinal de entrada de forma a poder transportar o sinal. Esta onda de transporte (**carrier wave**) tem, usualmente, uma frequência muito maior que o sinal de entrada, transmitindo a informação através do espaço por via de uma onda eletromagnética. Isto também permite que se transmita a diferentes frequências, de forma a que se possa partilhar o meio de transmissão, através de **multiplexagem de divisão de frequência**. Para tal, geralmente usam-se duas formas de modulação, muito importantes para a transmissão de sinais: **modulação por frequência** (habitualmente conhecida por FM, do inglês *frequency modulation*) e **modulação por amplitude** (habitualmente conhecida por AM, do inglês *amplitude modulation*).

Os **sinais sinusoidais** têm as suas formas definidas pela expressão $s(t) = A * \sin(2\pi * f * t + \text{fase})$. A **fase** do sinal consiste na amplitude com que o sinal inicia a sua propagação. Para manipular estes sinais, vulgarmente adicionam-se componentes (outras ondas de várias frequências).

Regeneração de sinais por modulação de amplitude de pulsos (PAM)

Quando recebemos um sinal analógico e pretendemos convertê-lo para digital temos de ter um padrão que nos permita reconhecer níveis de amplitude de forma a que nós possamos codificar o sinal analógico numa gama de símbolos discretos. Para tal usamos sistemas de codificação como o **NRZ unipolar**. O NRZ unipolar estabelece um valor mediano (v) entre a amplitude máxima da onda analógica e 0V, e codifica como '1' todos os pontos que passam para além de v e como '0' os que não passam. A Figura 11.2 mostra uma aplicação desta codificação.



transmissão dos sinais
transmissor
canal de transmissão

recetor

periódicos
frequência do sinal

moduladas
carrier wave

multiplexagem de divisão de frequência, modulação por frequência
modulação por amplitude
sinais sinusoidais
fase

NRZ unipolar

figura 11.2

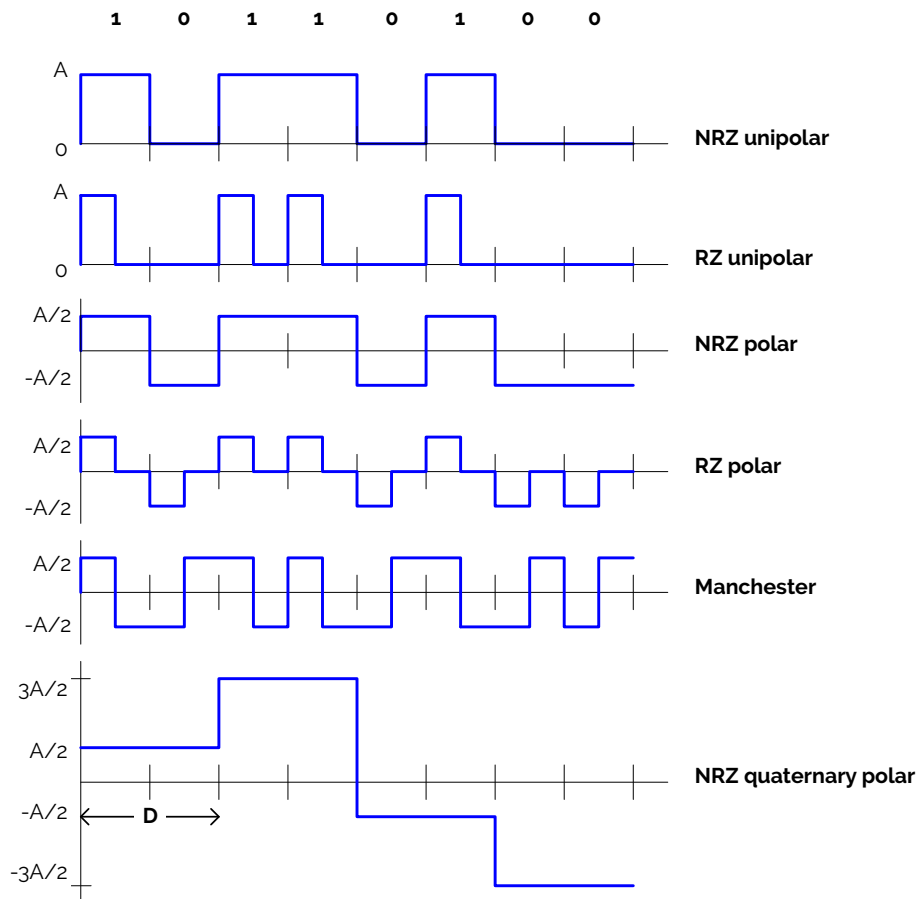
Como podemos ver na Figura 11.2, na codificação foram produzidos dois erros, que mais tarde, irão produzir, decerto, muito más consequências. Isto acontece porque, por si, há uma **probabilidade de erro** associada aos sistemas binários, no que toca à produção de codificações de sinais analógicos. Podendo aumentar o número de amostras e tentar corrigir o erro, outras opções são experimentar novas abordagens de codificação de sinais, isto é, novos algoritmos.

probabilidade de erro

Assumindo que o meio de propagação não introduz qualquer distorção e o ruído é Gaussiano (isto é, aditivo, branco, sem significado e independente do sinal), o sinal recebido no instante de tempo de amostragem é $y(t_k) = a_k + n(t_k)$, considerando que não há interferência entre símbolos (ISI). Se $y(t_k) > v$ o recetor decide '1', caso contrário decide '0'. Como podemos imaginar, o ruído pode causar fortes distúrbios e más decisões, pelo que a probabilidade do erro é de $P_e = P_0P_{e0} + P_1P_{e1}$, onde P_0 e P_1 são as probabilidades de '0' e de '1', respetivamente, P_{e0} é a probabilidade de erro quando '0' é transmitido e P_{e1} é a probabilidade de erro quando '1' é transmitido.

Vejamos então diferentes algoritmos de PAM, representados na Figura 11.3.

figura 11.3



Na Figura 11.3 começamos então com o algoritmo de NRZ unipolar. Este algoritmo codifica os sinais de forma uniforme, pelo que, em 0V e AV (daí unipolar) gera 0V quando é '0' e AV quando é '1'. Seguindo esta ordem de ideias, podemos considerar o **RZ unipolar** (*return-to-zero*) que quando está a '1', numa leitura, sobe para AV, seguido de 0V (regressa a zero), mantendo-se a 0V aquando do valor '0'. Estes mesmos algoritmos, nas suas versões polares, dão o **NRZ polar** (que é produto de uma translação do NRZ unipolar -1/2 para baixo, formando dois polos) e o **RZ polar** (que quando é '0' vai a -A/2V e volta a 0V e quando é '1' vai a A/2V e volta a 0V).

Finalmente temos o algoritmo de **Manchester** que faz uma transição descendente quando o valor é '1' e ascendente quando o valor é '0', e o algoritmo de **NRZ quaternary polar**, que toma como codificação, em quatro níveis, cada sequência de 2 bits, ocupada pelo que espaço denotado com D.

Comparemos então alguns dos algoritmos que estudámos e tiremos conclusões acerca de vantagens e desvantagens: primeiro, entre o unipolar e polar, temos que o unipolar tem sempre valores não-negativos e o polar varia entre positivo e negativo, o que é preferível, pois o consumo em termos energéticos é menor (consome uma média muito

RZ unipolar

NRZ polar

RZ polar

Manchester

NRZ quaternary polar

próxima de 0V). Entre o algoritmo RZ e NRZ temos que no NRZ o intervalo de frequência do sinal é pequena, sendo mais imune a desvios nos instantes ideais de amostragem, contra o RZ, que necessita de menos energia para a codificação, embora tenha mais níveis de voltagem para codificação. Finalmente entre o RZ polar e o Manchester, a energia necessária para o RZ polar é mais pequena, mas no Manchester há a simplificação dos requisitos de hardware, dado que não existe nível 0V e apenas se tem de detetar transições, melhorando também a questão do sincronismo no recetor.

A taxa de transmissão, considerando D como a duração de um símbolo (tal como representado na Figura 11.3), é igual a $r = 1/D$. Considerando também que M é o número de níveis de símbolos e T_b a duração de um bit, temos que, por exemplo, se $M = 2$, então $D = T_b$ e $r = 1/T_b$ (em bits/segundo). A **codificação M-ária** (M-ary coding), com $M = 2^n$ níveis, rebaixa a taxa de transmissão (ou **baudrate**) para $r = r_b / \log_2 M$.

**codificação M-ária
baudrate**

Considerando um símbolo polar com voltagem M tais que $a_k = \pm A/2, \pm 3A/2, \dots, \pm(M-1)A/2$ e assumindo que todos os símbolos M têm igual probabilidade, então a probabilidade de erro $P_e = (1/M)(P_{e0} + P_{e1} + \dots + P_{eM-1})$. Para um $M = 2$ - caso extremo - temos que ambos os símbolos terão uma probabilidade de erro igual ao caso binário anteriormente referido. Num Código de Gray, as combinações são representadas por níveis de símbolos consecutivos que só mudam num bit. Considerando um código desta génese e um SNR (sigla de *Signal to Noise Ratio*), a probabilidade de erro P_{be} fica aproximadamente igual a $P_e / \log_2 M$.

© Frank Gray

Sistema de modulação de códigos de pulso (PCM)

A transmissão digital de sinais analógico necessita de um **conversor analógico-digital** (ADC) na origem e um **conversor digital-analógico** (DAC) no destino. Na origem, da parte do conversor analógico-digital, os sinais analógicos são: amostrados com uma frequência $f_s \geq 2W$, onde W é a largura de banda efetiva do sinal; arredondados a q níveis de voltagem; e codificados em palavras com v símbolos, cada um com M níveis (símbolos M-ários).

**conversor analógico-digital,
conversor digital-analógico**

Assumindo que $|x(t)| \leq z$, a diferença entre níveis de arredondamento é de z/q e o máximo erro de arredondamento é $z/(2q)$. Neste caso, $q = M^v$ e $v = \log_M q$, com v símbolos e a taxa de transmissão é de $v * f_s$ baud. Diminuir a distorção provocada pelo arredondamento implica aumentar o número de níveis para o mesmo z , consequentemente, aumentar o baudrate.

Por exemplo, vejamos dois casos: no caso da telefonia, sendo $f_s = 8000$ amostras/s, com um $q = 256$ níveis, $M = 2$, $v = 8$, temos que o bit rate é de 64 Kbps, dado que $v * f_s = 8 * 8000$; num caso em que consideramos um sinal analógico $x(t)$ limitado numa amplitude tal que $|x(t)| < 5$ V e com uma largura de banda efetiva de 6 MHz a ser convertido num stream binário PCM para transmissão, temos que o data rate mínimo que garante um erro de arredondamento não maior que 0.05 V é obtido da seguinte forma:

- o erro máximo de arredondamento é $10/2q$, onde q é o número de níveis possíveis de arredondamento;
- q deve ser a menor potência de base 2 (dado que é um sinal binário) maior ou igual a $10/(2*0.05) = 100$, pelo que, $q = 128$;
- precisamos de saber quantos níveis, v : $\log_2(128) = 7$ bits por amostra;
- a frequência de amostragem mínima, $f_s = 2W$: $2 * 6 \text{ MHz} = 12 * 10^6$ amostras/s;
- o data rate mínimo é de $7 * 12 * 10^6 = 84 \text{ Mbps}$.

E assim termina a disciplina de Fundamentos de Redes (a3s1) continuando o esquema do conteúdo programático através das disciplinas de Arquiteturas de Redes (a3s2) e Arquiteturas de Redes Avançadas (a4s1).

1. Redes de Computadores e a Internet

Transmissão de informação e introdução ao protocolo IP	2
Endereçamento IP (versão 4)	3
Máscaras de Rede (IP versão 4)	5
Datagramas IP	5
Address Resolution Protocol (ARP)	6
Default gateway	7
Fragmentação e montagem IP	10
Sub-redes	11

2. Camada de Ligação (Data Link Layer) - I

Endereçamento em LANs (endereço IEEE)	12
Protocolo Ethernet (IEEE 802.3)	13

3. Redes de Área Locais Virtuais (VLAN)

Configuração de VLAN em switches	19
--	----

4. Spanning Tree Protocol (STP)

Estrutura de dados (árvore abrangente)	21
Ligação de árvores abrangentes em LANs	22

5. Protocolos de Acesso ao Meio

Topologias de rede	30
Token Ring (IEEE 802.5)	31
Wireless (IEEE 802.11)	33

6. Camada de Rede e endereçamento IP versão 6

Endereçamento privado	36
Network Address Translation (NAT)	36
Dynamic Host Configuration Protocol (DHCP)	37
Endereçamento IP versão 6 (IPv6)	39

7. Detecção e Controlo de Erros

Stop and Wait (SW)	41
Go-Back-N (GB-N)	42
Selective Repeat (SR)	43
Detecção de erros	44
Código de verificação de paridade	44
Cyclic-Redundancy Check (CRC)	45

8. Encaminhamento em redes IP

Estrutura hierárquica de redes	45
Distance vector versus link state	47
Protocolos distance vector	47
Problema da contagem para infinito e a separação de horizontes	49
Routing Information Protocol (RIP)	50

9. Camada de Transporte

User Datagram Protocol (UDP)	53
Transmission Control Protocol (TCP)	54
Estimativas de RTT e timeout do TCP	57
TCP como um motor de transmissão fiável de dados	57
Recuperação de erros em TCP: Go-Back-N ou Selective Repeat?	61
Controlo de fluxo em transmissões TCP	62
Controlo de congestionamento TCP	63

10. Camada de Aplicação

Aspetos-chave sobre aplicações	67
File Transfer Protocol (FTP)	68
Trivial File Transfer Protocol (TFTP)	70
Domain Name System (DNS)	72
HyperText Transfer Protocol (HTTP)	75
Correio eletrónico (e-Mail)	80

11. Camada Física

Sistemas de transmissão	82
Regeneração de sinais por modulação de amplitude de pulsos (PAM)	83
Sistema de modulação de códigos de pulso (PCM).....	85

Apontamentos de Fundamentos de Redes

2ª edição - novembro de 2016

fr

Autor: Rui Lopes

Fontes bibliográficas: Computer Networking - A Top-Down Approach, KUROSE, James F. et ROSS, Keith W., 6th edition, Addison-Wesley; Computer Networks, TANENBAUM, Andrew S. et WETHERALL, David J., 5th edition, Pearson

Outros recursos: Notas das aulas da disciplina de Fundamentos de Redes do ano letivo de 2015/2016 e 2014/2015.

Agradecimentos: professora Susana Sargento e professor António Nogueira

Todas as ilustrações gráficas são obra de Rui Lopes e as imagens são provenientes das fontes bibliográficas divulgadas.



apontamentos

© Rui Lopes 2016 Copyright: Pela Creative Commons, não é permitida a cópia e a venda deste documento. Qualquer fraude será punida. Respeite os autores e as suas marcas. Original - This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/deed.en_US.