TÉCNICO LISBOA

Engenharia Eletrotécnica e de Computadores

Redes de Computadores e Internet

Apontamentos sobre RCI

(Alguns tópicos)

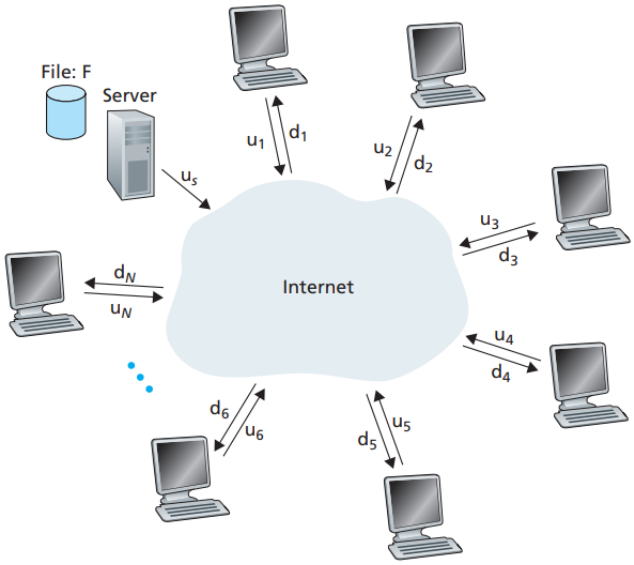


Imagem: *Illustrative file distribution problem* [*Kurose & Ross* 2017]

Autores:
João Gonçalves : 99995
↪ jrazevedogoncalves@tecnico.ulisboa.pt
Teresa Nogueira : 100029
↪ maria.teresa.ramos.nogueira@tecnico.ulisboa.pt

"*Let us reflect on the brief life of a bit.*"

Fevereiro 2023

Índice

1. Introdução às Redes de Computadores e à Internet

↳ Conceitos fundamentais

→ Redes de Computadores

Uma rede de computadores pode ser vista como um grupo de sistemas de computação conectados por meio de canais de comunicação em prole da comunicação e partilha de recursos de uma larga gama de utilizadores.

1. **Sistemas terminais** (*end systems or hosts*)—e.g., PCs, servidores, quintas de servidores (server farms), telemóveis, laptops, frigoríficos, termostatos, etc.
2. **Canal de comunicação** (*peer-to-peer*, difusão)—abstração de comunicação construida sobre um meio físico ou wireless: fios de cobre (e.g., RJ11, RJ45), cabos coaxiais (opção bastante isolada eletromagneticamente), fibra ótica (alta velocidade, opção imune a interferência eletromagnética), rádio nas suas múltiplas variantes...
3. A interligação supramencionada é efetuada via **comutadores**, que transitam informação entre canais adjacentes.
4. Exemplos de **serviços** de comunicação e partilha de recursos: encaminhamento de dados, garantia de entrega dos dados, autenticação, conversão entre nomes e endereços, etc.

→ Internet

A *Internet* é uma "rede de redes", global e hierarquizada, que providencia uma variedade de informação e serviços de comunicação (e.g., **aplicações distribuidas**) através da interconexão de redes que fazem uso de protocolos (de comunicação **standardizados**: abertos, sancionados por uma entidade competente (IETF), mas de adesão voluntária).

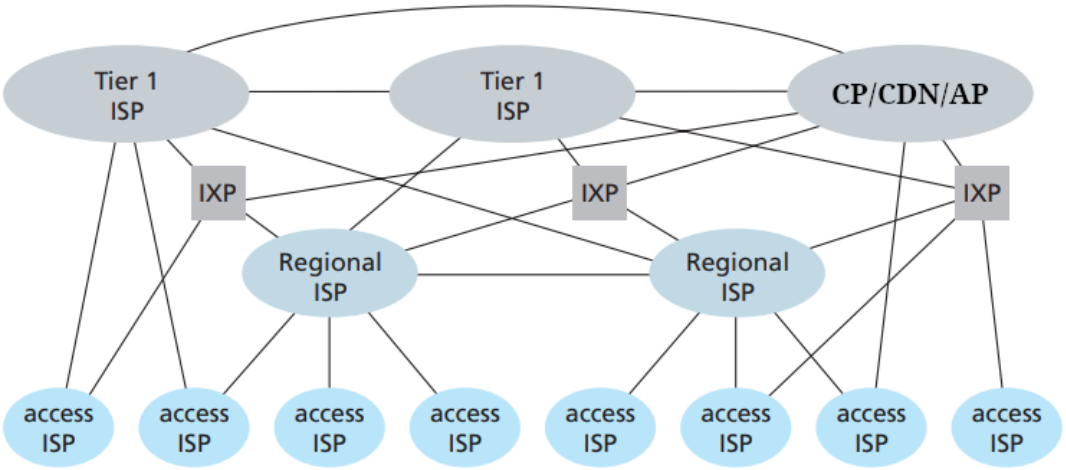


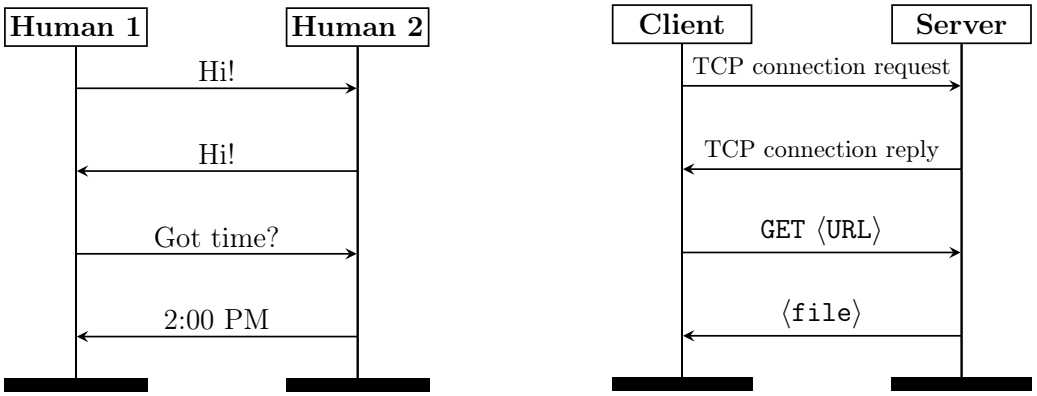
Fig. 1: Interconexão de ISPs. "In summary, today's Internet—a network of networks—is complex, consisting of a dozen or so tier-1 ISPs and hundreds of thousands of lower-tier ISPs. The ISPs are diverse in their coverage, with some spanning multiple continents and oceans, and others limited to narrow geographic regions. The lower-tier ISPs connect to the higher-tier ISPs, and the higher-tier ISPs interconnect with one another [*peering*, i.e., conectam as suas redes (settlement-free) de forma a que o tráfego comum passe pela conexão direta invés de por um ISP *upstream*]."[1]

CP – Content Provider CDN – Content Distribution Network AP – Application Provider

Nota: IXPs (Internet Exchange Points) são "pontos de encontro" para ISPs fornecidos por *third-parties* que dispõem de infraestruturas independentes com os seus próprios **comutadores**.

⋆ Protocolos

Consideremos agora outra *buzzword* importante em Redes de Computadores: *protocolo*.



(a) Protocolo "humano". (b) Protocolo de rede de computador[1].

Fig. 2: Exemplos genéricos de protocolos.

Protocolo

"A **protocol** defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event."[1]

Nota (no contexto computacional): "A *protocol* is an agreement about the packets exchanged by communicating programs and what they mean. A protocol tells how packets are structured—for example, where the destination information is located in the packet and how big it is—as well as how the information is to be interpreted."[2]

→ Breve História das Redes de Computadores e da Internet

1940: Máquina de Turing, primeira noção de um dispositivo computacional com comunicação.
1960: Mainframes começam a ser utilizadas em universidades e instituições militares para partilhar recursos e comunicar entre si.
1969: ARPANET, primeira rede de computadores, desenvolvida pelo Departamento de Defesa dos Estados Unidos com o objetivo de conectar instituições de pesquisa.
1972: Ray Tomlinson cria o primeiro programa de correio eletrónico, expandindo as funcionalidades da ARPANET além da simples troca de arquivos.
1974: Vint Cerf e Robert Kahn apresentam o protocolo TCP (Transmission Control Protocol), base para a comunicação na Internet.
1983: A ARPANET adota o protocolo TCP/IP, sedimentando as bases para a Internet moderna e interconexão global de redes.
1989: Tim Berners-Lee, no CERN, propõe a World Wide Web, um sistema global de documentos interligados por hiperligações e acessíveis pela Internet.
1990: ARPANET é oficialmente desativada, dando lugar à Internet atual.
1993: Mosaic, primeiro navegador gráfico da WWW, desenvolvido por Marc Andreessen e Eric Bina, populariza a Internet e atrai um público mais amplo.
1994: O protocolo HTTP é desenvolvido por Tim Berners-Lee, facilitando a troca de informações na World Wide Web.

⋮

[1]São exemplos de protocolos: TCP, UDP, HTTP/HTTPS; que veremos mais adiante.

↳ Multiplexagem e comutação

→ Circuit Switching

Circuit Switching

"In circuit-switched networks, **the resources needed along a path** to provide for communication between the end systems **are *reserved*** for the duration of the communication session between the end systems."[1]

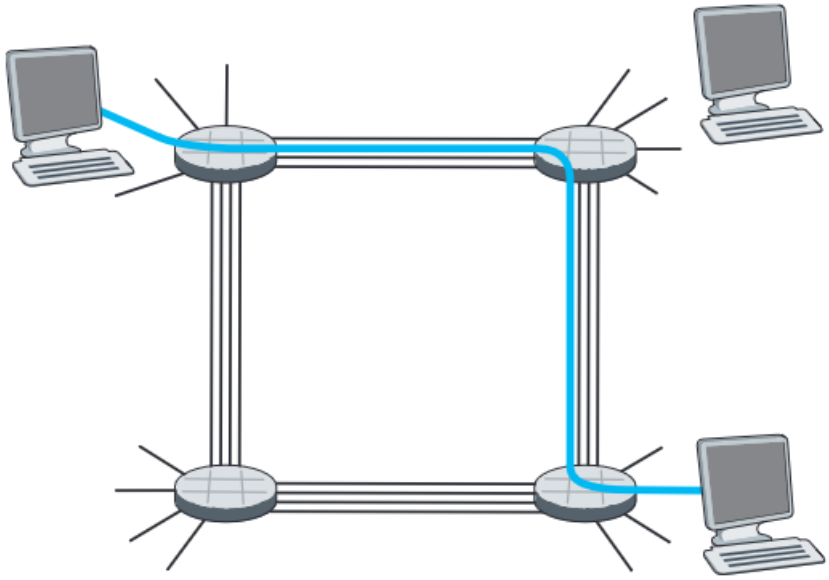


Fig. 3: Circuit Switching: "In this network, the four circuit switches are interconnected by four links. Each of these links has four circuits, so that each link can support four simultaneous connections. The hosts (for example, PCs and workstations) are each directly connected to one of the switches. **When two hosts want to communicate, the network establishes a dedicated end-to-end connection between the two hosts.** Thus, in order for Host A to communicate with Host B, the network must first reserve one circuit on each of two links. **In this example, the dedicated end-to-end connection uses the second circuit in the first link and the fourth circuit in the second link.**"[1]

A coordenação entre canal-subcanal de entrada e saída (referente à última frase da legenda da Fig. 3) é realizada através do uso de **tabelas de expedição** dinâmicas, que veremos mais adiante.

Recorrendo novamente à Fig. 3, cada canal possui quatro circuitos, a velocidade transmissão será um quarto da capacidade total do canal:

> "Because each link has four circuits, for each link used by the end-to-end connection, the connection gets one fourth of the link's total transmission capacity for the duration of the connection."[1]

$$R_{\text{trans}} = \frac{R_{\text{link}}}{N} \text{ [bps]}$$

Onde N é o número de circuitos por canal (*circuits* por *link*).

⋆ **Multiplexagem FDM (*frequency-division multiplexing*)**

FDM

"With FDM, **the link dedicates a frequency band to each connection** for the duration of the connection"[1]

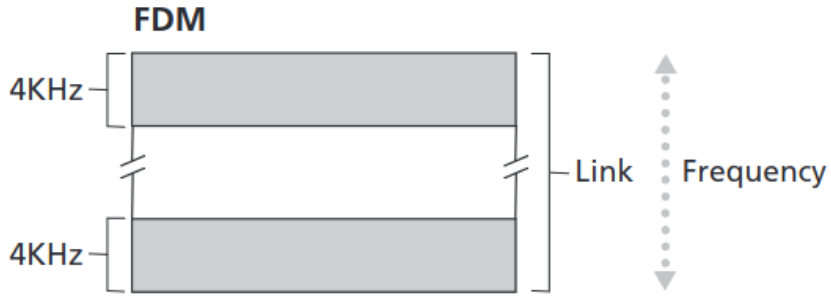


Fig. 4: Vizualição das partições da banda de transmissão na multiplexagem de frequência

⋆ **Multiplexagem TDM (*Time-division multiplexing*)**

TDM

"For a TDM link, time is divided into frames of fixed duration, and each frame is divided into a fixed number of time slots. When the network establishes a connection across a link, the network dedicates one time slot in every frame to this connection."[1]

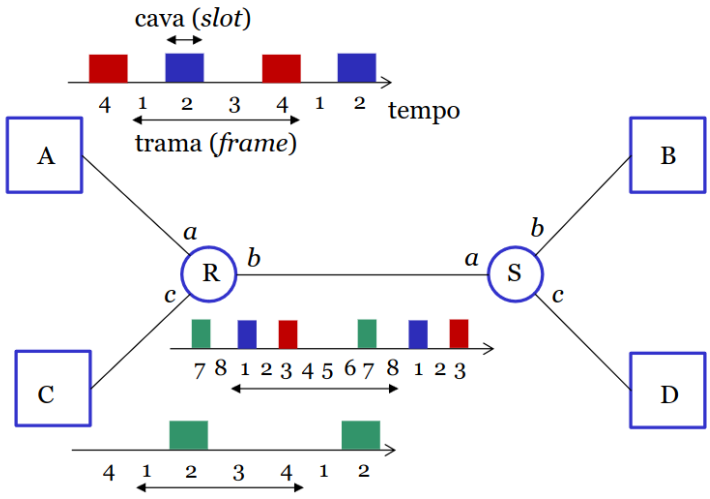


Tabela de expedição em R

Entrada		Saída	
canal	cava	canal	cava
a	2	*b*	1
a	4	*b*	3
c	2	*b*	7

Fig. 5: Time Division Mulplexing com respetiva tabela de expedição.

A informação é dividida no tempo em *slots* (cavas), o conjunto de *slots* origina uma estrutura denominada *frame* (trama) de natureza periódica. A comutação entre canais é realizada através de tabelas de expedição que estabelecem associações entre canal e subcanal de entrada e saída:

→ Na Fig. 5, a informação a *verde* comuta do canal *c* para o *b*, sofrendo também uma comutação do subcanal 2 para o 7 (*vide* tabela de expedição e a numerização dos slots). O mesmo processo verifica-se para a informação que comuta do canal *a* para o canal *b*.

⋆ Multiplexagem Determinística

O processo de *circuit switching* é um método multiplexagem determinística (transmissão constante, certa, "When the network establishes the circuit, it also reserves a constant transmission rate in the network's links"[1])

▲ Multiplexagem
- Divisão de um canal em sub-canais de capacidades fixas (**TDM**, **FDM**, WDM, CDM).

▲ Comutação (*switching*)
- Circuito: concatenação de sub-canais ao longo de um caminho.
- Tabelas de expedição (fowarding table): associação entre pares de entrada (canal, sub-canal) e pares de saída (canal, sub-canal).

▲ Circuitos dinâmicos:
- Estabelecimento do circuito: atribuições de sub-canais em cada canal de um caminho e preenchimento das tabelas de expedição.
- Terminação do circuito: remoção das atribuições e limpeza das tabelas.
- Bloqueio se um canal não tiver um sub-canal disponível.

→ Packet Switching

Packet Switching

"**The source breaks long messages into smaller chunks of data known as *packets***. Between source and destination each packet travels through communication links and ***packet switches***—a router takes a packet arriving on one of its attached communication links and forwards that packet onto another one of its attached communication links"[1]

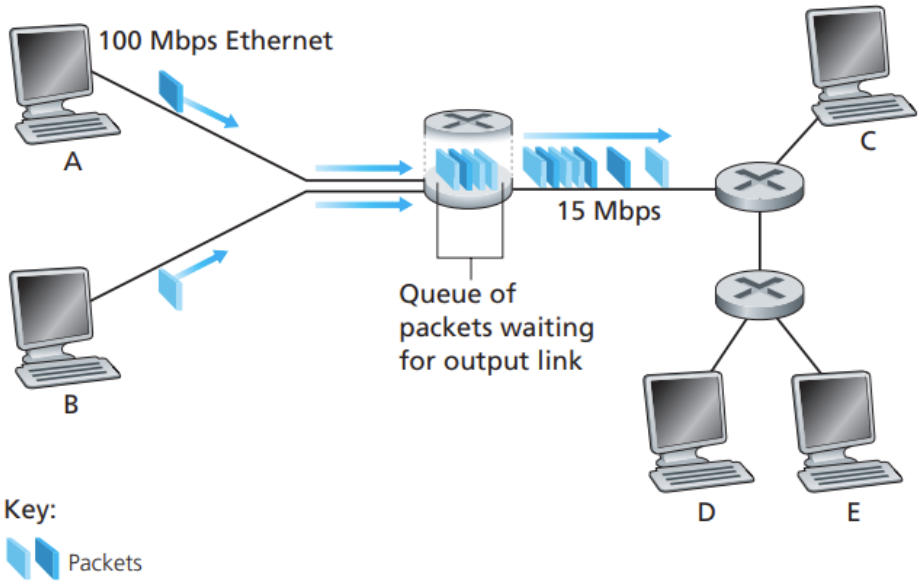


Fig. 6: Packet Switching

*"Finally, there are two different ways to pronounce the word router, either as "rootor" or as "rowter," and people waste a lot of time arguing over the proper pronunciation [Perlman 1999]."[1]

Estes pacotes viajam com um ritmo igual ao ritmo total de transmissão do canal. Neste sentido, supondo o envio de um pacote de dimensão L bits por um canal com capacidade R bits/sec, o intervalo de tempo de transmissão é dado por:

$$\boxed{d_{\text{trans}} = \frac{L}{R} \text{ [s]}}$$

⋆ Store and Forward Transmission

Store and Forward Transmission

"Store-and-forward transmission means that **the packet switch *must* receive the entire packet** before it can begin to transmit the first bit of the packet onto the outbound link."[1]

Pressupondo a transmissão de um pacote até um *router* (*packet switcher*) que aplica *Store and Forward Transmission*, a subsequente comutação e transmissão para outro canal é apenas executada quando o *router* possuir o pacote completo. Assim os bits já recebidos do pacote sofrem *buffering* (são armazenados em fila de espera) até o *router* possuir o pacote integral e proceder à sua transmissão.

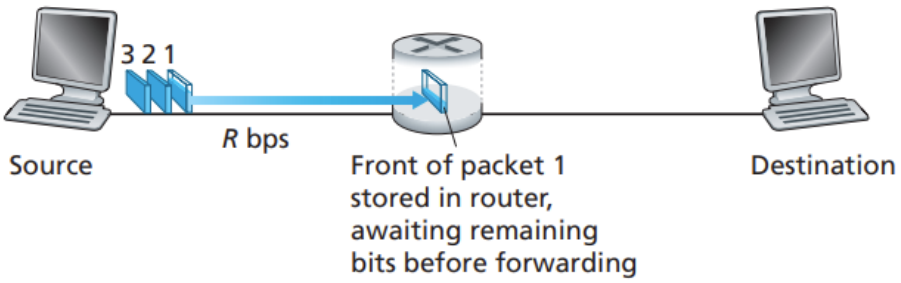


Fig. 7: Store and Forward transmission

O tempo de espera resultante do processo de *buffering* denomina-se de ***queuing delay***, d_{queue}. Este *delay* é aleatório e está dependente do nível de **congestionamento** do canal (a taxa de chegada de pacotes a um comutador é superior à sua capacidade). Por outro lado o espaço da linha de espera é finito. Num caso extremo de congestionamento a linha de espera poderá encontrar-se totalmente preenchida, pacotes acabados de chegar poderão ser parcialmente ou totalmente perdidos, ocorre ***packet loss***.

⋆ Forwarding Tables and Routing Protocols

Forwarding Tables

"(...) Each router has a forwarding table that maps destination addresses (or portions of the destination addresses) to that router's outbound links. **When a packet arrives at a router, the router examines the address and searches its *forwarding table***, using this destination address, **to find the appropriate outbound link**."[1]

As tabelas de expedição (*forwarding tables*, localizadas nos cabeçalhos dos pacotes) são elaboradas automaticamente com recurso a ***routing protocols***:

> "A ***routing protocol*** may, for example, determine the shortest path from each router to each destination and use the shortest path results to configure the forwarding tables in the routers."[1]

⋆ Multiplexagem Estatística

O processo de *packet switching* é um método de multiplexagem estatística—decorre da partilha assíncrona de um canal entre pacotes e obedece à política *first come first served* (a transmissão de pacotes é realizada por ordem de chegada). A qualidade estatística refere-se ao caráter aleatória do congestionamento e *rate* de transmissão:

▲ Multiplexagem
- Mensagens e *data streams* são divididas em pacotes (*packages*) de pequena dimensão.
- Partilha assíncrona (no tempo) de um canal entre pacotes.
- Cabeçalhos distinguem os pacotes.
- Os cabeçalhos (*headers*) dos pacotes contêm informação que permite o seu posterior reagrupamento nos dados originais.

▲ Comutação
- *Store-and-forward*: cada mensagem é recebida na totalidade antes de ser expedida.
- As tabelas de expedição estabelecem a localização do subsequente *outbound-link* através da associação entre terminais de entrada e saída.

▲ **Equidade** na partilha dos recursos de transmissão e comutação (*switching*).

▲ **Congestionamento** se a taxa de chegada de mensagens a um comutador for superior à capacidade deste, conduzindo a atrasos e, possivel perdas.

Nota: Ao não executar a partição de dados em pacotes de menor tamanho a transmissão fica sujeita a períodos de **inequidade** (*starvation*): quando uma mensagem longa impede o despacho de mensagens curtas.

→ Circuit Switching vs. Packet Switching

Circuit Switching	Packet Switching
Ideal para aplicações que exigem comunicação confiável/determinística, como as redes telefónicas.	Garante flexibilidade e eficiência para redes com um elevado número de utilizadores.
Fornece uma qualidade de serviço consistente.	Experiência de comunicação imprevisível, graças à política de *first come first served*.
Requer um caminho dedicado durante toda a sessão de comunicação, mesmo durante períodos de inatividade.	Mais simples, mais eficiente (em termos de recursos), e menos dispendiosa de implementar

Tab. 1: Pros & cons dos tipos de comutação abordados.

→ Virtual Circuits

Virtual Circuits

"Virtual circuits provide a logical, **pre-determined path within a packet-switched network**, such as Frame Relay, Asynchronous Transfer Mode (ATM), and Multiprotocol Label Switching (MPLS) networks. Virtual circuits use statistical multiplexing, which is an intrinsic feature of packet-switched networks, allowing multiple virtual circuits to share the same physical links and resources, leading to better utilization of network resources and increased efficiency." [?]

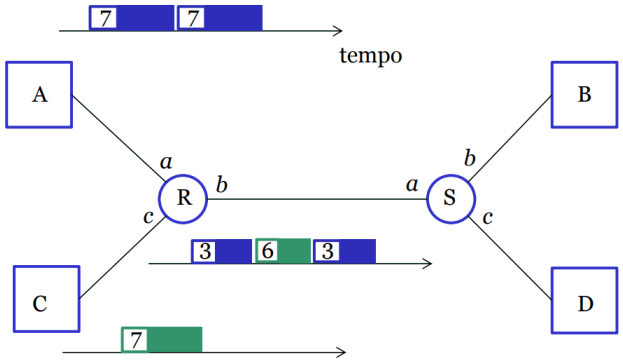


Tabela de expedição em R

Entrada		Saída	
porta	VCI	porta	VCI
a	7	*b*	3
c	7	*b*	6

VCI – *Virtual Circuit Identifier*

Fig. 8: Circuitos virtuais.[3]

While virtual circuits do not reserve a fixed bandwidth like traditional circuit-switched networks, they can offer certain advantages such as:

- **Established path**, which may lead to more consistent performance compared to purely connectionless packet switching.
- **In-order delivery** of packets, as they follow the same pre-determined path.

By allowing multiple virtual circuits to share the same physical links and resources, virtual circuits provide greater flexibility and efficiency compared to traditional circuit-switched networks.

↳ Débitos e atraso

→ Tipos de atraso (*delay*)

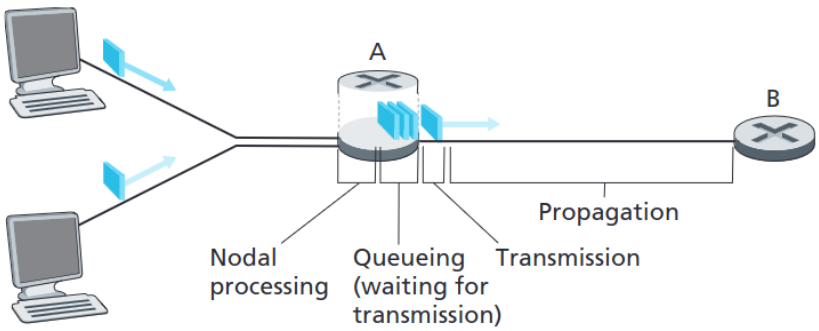


Fig. 9: Tipos de atraso

$$d_{\text{switch}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

Processing Delay (Atraso de Comutação): $d_{\textbf{switch}}$

"The time required to examine the packet's header and determine where to direct the packet is part of the ***processing delay***. The processing delay can also include other factors, such as the time needed to check for bit-level errors in the packet that occurred in transmitting the packet's bits from the upstream node to router."[1]

Queue Delay (Atraso de fila de espera): $d_{\textbf{queue}}$

"At the queue, the packet experiences a ***queuing delay*** as it waits to be transmitted onto the link. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. "[1]

Nota: o atraso de fila de espera é dependente da intensidade de tráfego, La/R, em que a é dado em [packets/s]. Se $La/R > 1$ o *rate* de chegada de pacotes excede o *rate* de transmissão, "the queue will tend to increase without bound and the queuing delay will approach infinity!"[1]

Transmission Delay (Atraso de Transmissão): $d_{\textbf{trans}}$

"The ***transmission delay*** is L/R. This is the amount of time required to push (that is, transmit) all of the packet's bits into the link. "[1]

Propagation Delay (Atraso de Propagação): $d_{\textbf{prop}}$

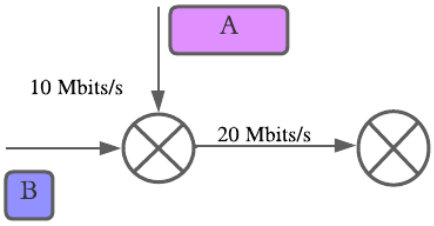
"The ***propagation delay*** is the distance between two routers divided by the propagation speed. That is, the propagation delay is d/s, where d is the distance between router A and router B and s is the propagation speed of the link."[1]

[§]**Nota:** Leia-se L [bits] como *length* do pacote, e R [bps] como o ritmo de transmissão/capacidade.

→ Exemplos

⋆ Comutação de mensagens *versus* comutação de pacotes

Considere um comutador com duas linhas de entrada, cada uma com capacidade 10 Mbit/s, e uma linha de saída com capacidade 20 Mbit/s. Na primeira linha chegam ao comutador mensagens do tipo A contendo 10 MB cada e na segunda linha chegam mensagens do tipo B contendo 1 KB cada.

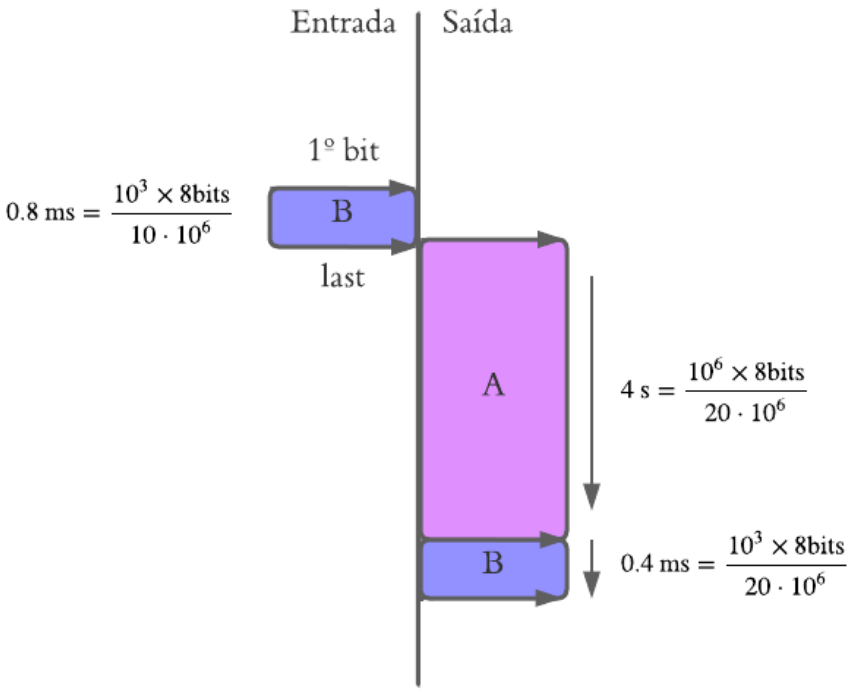


Comutação de mensagem: Toda a mensagem é recebida e logo a seguir transmitida se a linha de saída estiver desocupada.

Comutação de pacotes: A mensagem é segmentada em pacotes. Cada pacote é recebido e transmitido por ordem de chegada.

▲ Assumindo multiplexagem de mensagens, qual é, no pior caso, o atraso a que está sujeita uma mensagem do tipo B, medido desde o momento em que a mensagem chega ao comutador até que o seu último bit é transmitido na linha de saída?

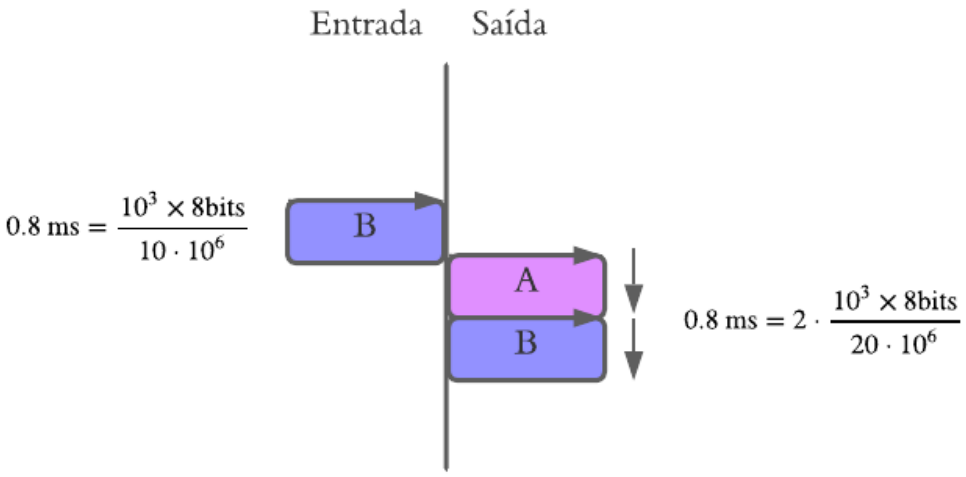
[A] : O **pior caso** denota a ocupação da linha de saída pela mensagem A. Desta forma, a transmissão de uma mensagem de tipo B terá partida apenas quando toda a mensagem A for transmitida para o terminal de chegada (*store and forward*):



A mensagem B chega imediatamente antes do início da transmissão de A, e aguarda a sua transmissão integral, por fim, ocupa a linha de saída (não é considerado o atraso proveniente da linha de entrada):

$$\therefore \text{ Atraso Total} = 0.4\text{ ms} + 4\text{ s} = \mathbf{4.0004\text{ s}}$$

▲ Assumindo multiplexagem de pacotes, cada um com 1 KB, qual é agora o atraso a que está sujeita uma mensagem do tipo B?



[A] : Assumindo multiplexagem por pacotes, o atraso total será dado pelo delay da transmissão dos pacotes A e B:

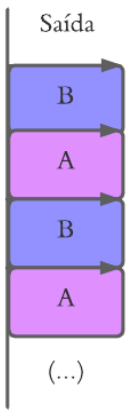
$$\therefore \text{Atraso Total} = 0.4\text{ ms} + 0.4\text{ ms} = \mathbf{0.8\text{ ms}}$$

Passa a existir equidade na partilha de recursos de transmissão e comutação (a mensagem B não monopoliza a linha de saída).

▲ O que pode dizer relativamente ao atraso sofrido pelas mensagens do tipo A num caso e no outro?

[A1] : Assumindo multiplexagem por mensagem para transmissão da mensagem A o atraso para o pior caso calculado será idêntico ao já vizualizado na primeira alínea, 4.0004 s, já que apenas se verifica uma troca na ordem de transmissão. Assim a mensagem A terá de esperar 0.4 s provenientes da ocupação da linha de transmissão pela mensagem de tipo B e subsequentemente terá de aguardar 4 s até que todos os seus bits tenham sido transmitidos.

[A2] : Supondo agora comutação de pacotes:



Neste cenário os pacotes de A e de B saem intercalados (*interleaved*). Assim, a mensagem A—composta por 10000 pacotes—tem um atraso de:

$$d_{\text{total}} = 10000 \cdot \left(2 \cdot \frac{10^3 \times 8}{20 \cdot 10^6}\right) = 10^5 \cdot 0.8\,\text{ms} = \mathbf{80\,s}$$

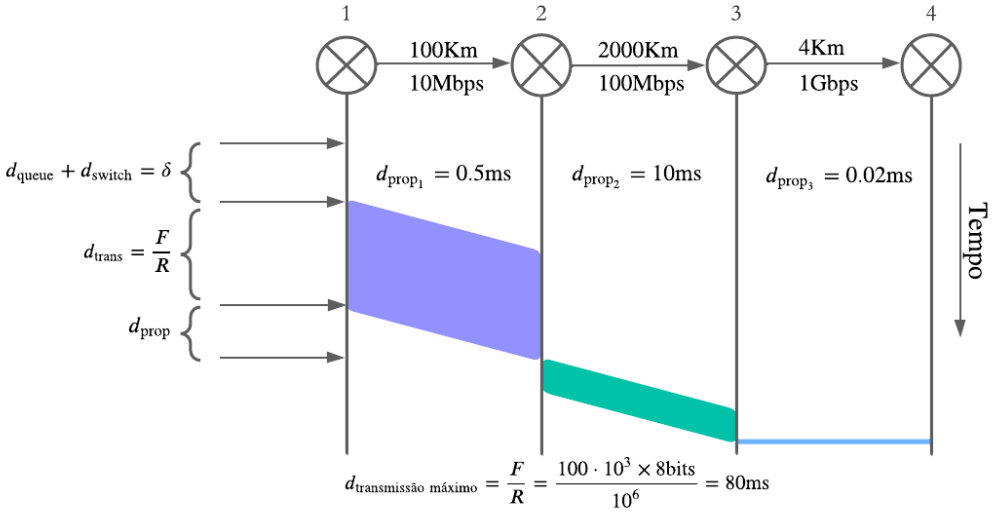
Visto que cada pacote tem um atraso de 0.8 ms (sempre que um pacote de uma mensagem de tipo A entra no canal, este encontra-se ocupado por uma mensagem de tipo B).

⋆ Entrega de um ficheiro ao longo de um caminho

Considere o envio de um ficheiro contendo 100 KB através de um caminho composto por três troços de fibra óptica interligados por dois comutadores. O primeiro troço tem 100Km de comprimento e capacidade 10 Mbit/s; o segundo tem 2.000 Km de comprimento e capacidade 100 Mbit/s; e o terceiro tem 4 Km de comprimento e capacidade 1 Gbit/s. A velocidade de propagação da luz na fibra é de 200.000 km/s. É usada uma tecnologia de comutação de pacotes.

Nota: Embora seja referida comutação de pacotes, considera-se a dimensão do pacote deprezável face à dimensão do ficheiro

▲ Qual o atraso na entrega do ficheiro?



[A] : Invocando a fórmula do cálculo aproximado da latência na entrega de mensagens com N pacotes ($F = NL$):

$$d_{\text{total}} \approx \delta + \frac{F}{\min_i\{R_i\}} + \sum_{i=1}^{n-1} d_{\text{prop},i}$$

temos então, para $F = 100$ KB, um atraso: $d_{\text{total}} \approx \delta + 0.5 + 10 + 0.02 + 80 = 90.52$ ms $+ \delta$.

▲ Quais dos comutadores experimentam uma ocupação da sua fila de espera?

[A] : Nenhum, os ritmos são crescentes a jusante, i.e, o ritmo de entrada é sempre inferior ao ritmo de saída, o ritmo de crescimento da fila é negativo.

▲ Suponha agora que as capacidades dos dois primeiros troços são trocadas. Isto é: o primeiro troço fica com capacidade 100 Mbit/s e o segundo fica com capacidade 10 Mbit/s. Qual o atraso na entrega do ficheiro?

[A] : O atraso é idêntico ao calculado na primeira alínea. A fórmula é invariante à posição dos comutadores e limitada apenas pela menor capacidade ao longo de toda a transmissão, que continua a ser 10 Mbits/s.

▲ Qual a ocupação máxima da fila de espera à saída do primeiro comutador?

[A] : A fila de espera cresce a um ritmo de 100Mbps − 10Mbps = 90Mbps, durante 8 ms (tempo de transmissão assumindo um $R_{\text{entrada}} = 100$Mbps, como na alinea anterior).

$$\underline{\text{Tamanho da fila}} = 90\text{Mbps} \cdot 8\text{ms} = 720000 \text{ bits} = 90\text{KB} \implies \frac{90\text{KB}}{F} = 0.9 = 90\%$$

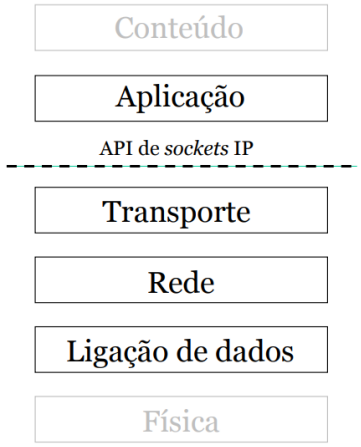
↳ Arquitetura em camadas

De modo a que a rede (de protocolos) tome uma estrutura, é necessário organizar os **protocolos** (tomamos uma abordagem em camadas—em que cada protocolo pertence a uma camada). Refletimos sobre os **serviços** que uma camada providencia à camada diretamente acima (**service-model** da camada).

> "(...) each layer provides its service by (1) performing certain actions within that layer and by (2) using the services of the layer directly below it."[1]

Uma camada pode ser implementada em *software*, em *hardware*, ou numa combinação de ambos; e é usual que os componentes integrantes da rede (e.g., sistemas terminais, comutadores, etc.) tenham partes do protocolo da camada *n distribuídos* entre si.

Este modelo (*protocol layering*) apresenta vantagens estruturais[2] e conceptuais que nos permitem chegar à *pilha de protocolos da Internet*:



- **Camada de Aplicação** – Protocolos dependentes da aplicação em causa (HTTP/HTTPS, SMTP, ...)
- **Camada de Transporte** – Controlo de erros e de fluxo extremo-a-extremo, controlo de congestionamento (transferência de dados entre processos; TCP, UDP, etc.)
  transport-layer packets → ***segments***
- **Camada de Rede**[3] – Encaminhamento e expedição global (dos *datagramas*)
  network-layer packets → ***datagrams***
- **Camada de Ligação** – Controlo de erros local, encaminhamento e expedição local, controlo de acesso ao meio (Ethernet, Wi-Fi, PPP, ...) *link-layer* → ***frames***

Fig. 10: Modelo simplificado da Internet—pilha de protocolos.

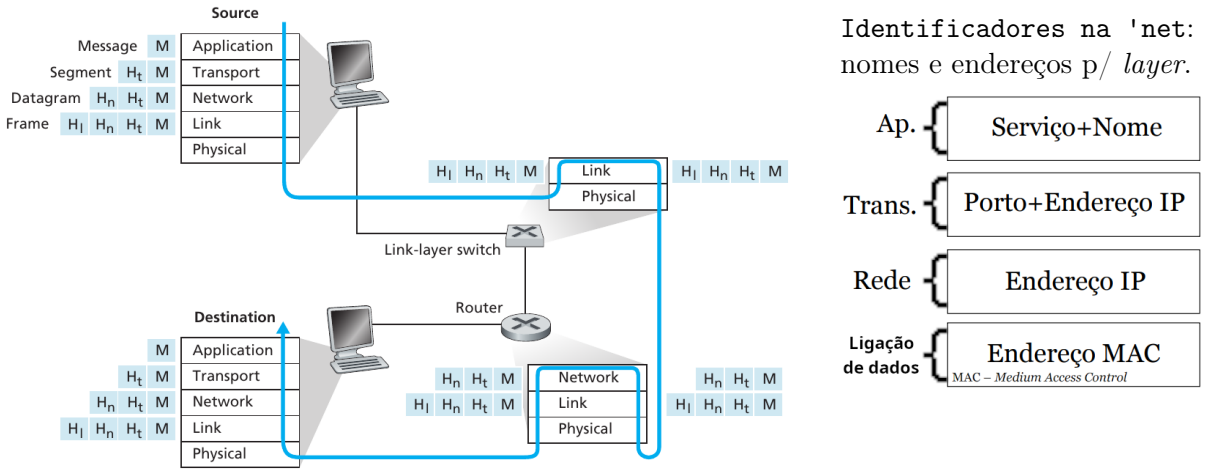


Fig. 11: Cada componente da rede dispõe de um conjunto de camadas, o que reflete a diferença entre as suas funcionalidades. **Encapsulamento:** o pacote da camada N é constituído pelo cabeçalho da camada N mais o pacote da camada $N+1$.

[2] "For large and complex systems that are constantly being updated, the ability to change the implementation of a service without affecting other components of the system [veja-se como modularidade] is another important advantage of layering."[1]

[3] "Although the network layer contains both the IP protocol and numerous routing protocols, it is often simply referred to as the IP layer, reflecting the fact that IP is the glue that binds the Internet together."[1]

[§] **Nota:** API – Application Programming Interface IP – Internet Protocol.

2. Camada de Aplicação

↳ Arquiteturas de aplicação de redes

Application Architecture

"The application architecture, on the other hand, is designed by the application developer and dictates how the application is structured over the various end systems. In choosing the application architecture, an application developer will likely draw on one of the two predominant architectural paradigms used in modern network applications: **the client-server architecture** or the **peer-to-peer (P2P)** architecture."[1]

→ *Client-server architecture*

Em arquiteturas cliente-servidor (*client-server architectures*) um servidor (*always-on host*) **providência os seus serviços a múltiplos clientes**: "A classic example is the Web application for which an always-on Web server services requests from browsers running on client hosts." onde **os clientes não comunicam diretamente entre si** (o servidor é responsável pela manutenção e comunicação entre clientes). Os servidores **possuem um IP fixo**, garantindo um fluxo de comunicação constante. Para garantir o processamento de um elevado tráfego de pedidos são utilizados *data centers* que albergam múltiplos *hosts*:

> "The most popular Internet services—such as search engines (e.g., Google, Bing, Baidu), Internet commerce (e.g., Amazon, eBay, Alibaba), Web-based e-mail (e.g., Gmail and Yahoo Mail), social networking (e.g., Facebook, Instagram, Twitter, and WeChat)—employ one or more data centers."[1]

→ *Peer-to-Peer (P2P)*

Em arquiteturas *peer-to-peer* (P2P) a comunicação é independente de servidores. A aplicação procura estabelecer contacto direto entre pares de *hosts* ligados intermitentemente designador por *peers*. Estes *host* residem nas máquinas dos utilizadores (e.g. casa, universidades, escritorios, ...). A comunicação P2P é dotada de *self-scalability*:

> "For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers."[1]

Client-server architecture	Peer-to-Peer (P2P)
O servidor é dependente de *data centers* que necessitam de manutenção e energia.	*Cost efficient*, não dependente de infraestruturas de larga escala.
Pagamento de *recurring interconnection and bandwidth costs* na transmissão de dados.	"Face challenges of security, performance, and reliability due to their highly decentralized structure."
Seguro e estável, face à arquitetura *peer-to-peer*	*Self-scalability*.

Tab. 2: Pros & cons dos tipos de arquitetura de aplicação abordados.

→ Comunicação entre processos

Processo (Process)

"In the jargon of operating systems, it is not actually programs but processes that communicate. **A process can be thought of as a program that is running within an end system**. When processes are running on the same end system, they can communicate with each other with **interprocess communication**. Processes on two different end systems communicate with each other by **exchanging messages across the computer network**."[1]

Uma aplicação de rede consiste na troca de mensagens entre pares de processos. Cada par possui um processo designado cliente e outro servidor, mediante a tarefa desempenhada no início da sessão de comunicação:

> "In the context of a communication session between a pair of processes, **the process that initiates the communication** (that is, initially contacts the other process at the beginning of the session) **is labeled as the client. The process that waits to be contacted to begin the session is the server**."

Um processo envia e recebe mensagens da rede através de uma interface designada *socket* que estabelece a ligação entre a camada de aplicação e a camada de transporte da qual o programador pouco controlo tem com exceção da escolha do protocolo em uso. Este tema será explorado na secção 2.x.

→ Endereçamento de processos

A identificação do processo de destino numa sessão de comunicação está dependente de dois identificadores:

⋆ **Endereço IP:** Identifica o *host* na internet.

"Internet addresses are binary numbers. They come in two flavors, corresponding to the two versions of the Internet Protocol that have been standardized. The most common is version 4 (IPv4); the other is version 6 (IPv6), which is just beginning to be deployed. IPv4 addresses are 32 bits long; because this is only enough to identify about 4 billion distinct destinations, they are not really big enough for today's Internet. For that reason, IPv6 was introduced. IPv6 addresses are 128 bits long."[2]

Exemplo: IPv4 – 193.136.128.169 IPv6 – 2001:690:21c0:a::150

⋆ **Porto:** Especifica o processo de chegada (a socket de chegada) dentro do *host*, já que este poderá estar a correr várias aplicações em simultâneo.

"Port numbers are the same in both IPv4 and IPv6: 16-bit unsigned binary numbers. Thus, each one is in the range 1 to 65,535 (0 is reserved)"[2]

→ Serviços fornecidos pela camada de transporte

Category	Description
Reliable Data Transfer	A transport-layer protocol guarantees that data sent by one end is correctly and completely delivered to the other end.
Throughput	A transport-layer protocol can provide a guaranteed available throughput at a specified rate, appealing to bandwidth-sensitive applications. Elastic applications can use any available throughput.
Timing	A transport-layer protocol can provide timing guarantees, ensuring data delivery within a specified time frame. This is essential for interactive real-time applications.
Security	A transport protocol can provide security services such as encryption, data integrity, and end-point authentication to protect data transmitted between processes.

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Smartphone messaging	No loss	Elastic	Yes and no

Fig. 12: Requisitos de algumas aplicações de redes.[1]

⋆ Transmission Control Protocol (TCP)

O protocolo TCP fornece um serviço orientado à sessão (connection-oriented) de transferência fiável (*reliable data transfer*) e entrega sequencial de pacotes. Serviços orientados à sessão envolvem um processo de *handshaking* entre cliente e servidor de modo a trocar informação de controlo e preparar ambos os lados para a transmissão de dados, esta transmissão é bidirecional. Possui um mecanismo de *congestion control* e **não fornece garantias de atraso nem de débito**.

⋆ User Datagram Protocol (UDP)

UDP é um protocolo de transporte *lightweight* que providência serviços minimos e não envolve processo de *handshaking* (é *connectionless*) **não possui uma transferência fiável nem garante uma entrega sequencial de pacotes**.

↳ The Web & HTTP

World Wide Web (WWW)

"The WWW is a global information space that consists of interconnected documents (Web pages) and resources accessible through the internet. The Web is built on top of the Hypertext Transfer Protocol (HTTP), which is a fundamental protocol used for transmitting and sharing data between web browsers (clients) and web servers."

→ HTTP overview

Hypertext Transfer Protocol (HTTP)

"HTTP is an application-layer protocol that enables the exchange of documents, images, videos, and other resources on the World Wide Web. It uses a client-server model, where the client initiates requests and the server responds with the requested resource or an error message."

1. **Request-Response Cycle**: Clients (usually web browsers) send HTTP requests to servers, which respond with the requested resource or an error message.
2. **HTTP Methods**: HTTP defines several methods, such as `GET`, `POST`, `PUT`, `DELETE`, and others, to specify the desired action to be performed on the requested resource.
3. **HTTP Status Codes**: Servers respond with HTTP status codes to indicate the outcome of an HTTP request. Common status codes include 200 (OK), 404 (Not Found), and 500 (Internal Server Error).

→ **Nota:** HTTP utiliza **TCP** como protocolo de transporte!

HTTP é dito um *stateless protocol*, dado que um servidor HTTP não retem informação acerca do utilizador:

> "If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier."[1]

"HTTP has evolved through several versions, each introducing new features and improvements:

- **HTTP/1.0:** The initial version of HTTP, supporting basic request-response communication and using non-persistent connections.
- **HTTP/1.1:** An updated version that introduced persistent connections, chunked encoding, and improved caching mechanisms.
- **HTTP/2:** A more recent version that offers significant performance improvements, such as multiplexing, header compression, and server push.
- **HTTP/3:** The latest version, currently under development, which aims to improve performance further by using the QUIC transport protocol instead of TCP." [**?**]

→ Persistência

Persistência

"(...) The application developer needs to make an important decision—should each request/response pair be sent over a separate TCP connection, or should all of the requests and their corresponding responses be sent over the same TCP connection? In the former approach, the application is said to use non-persistent connections; and in the latter approach, persistent connections."

⋆ Sessão não persistente

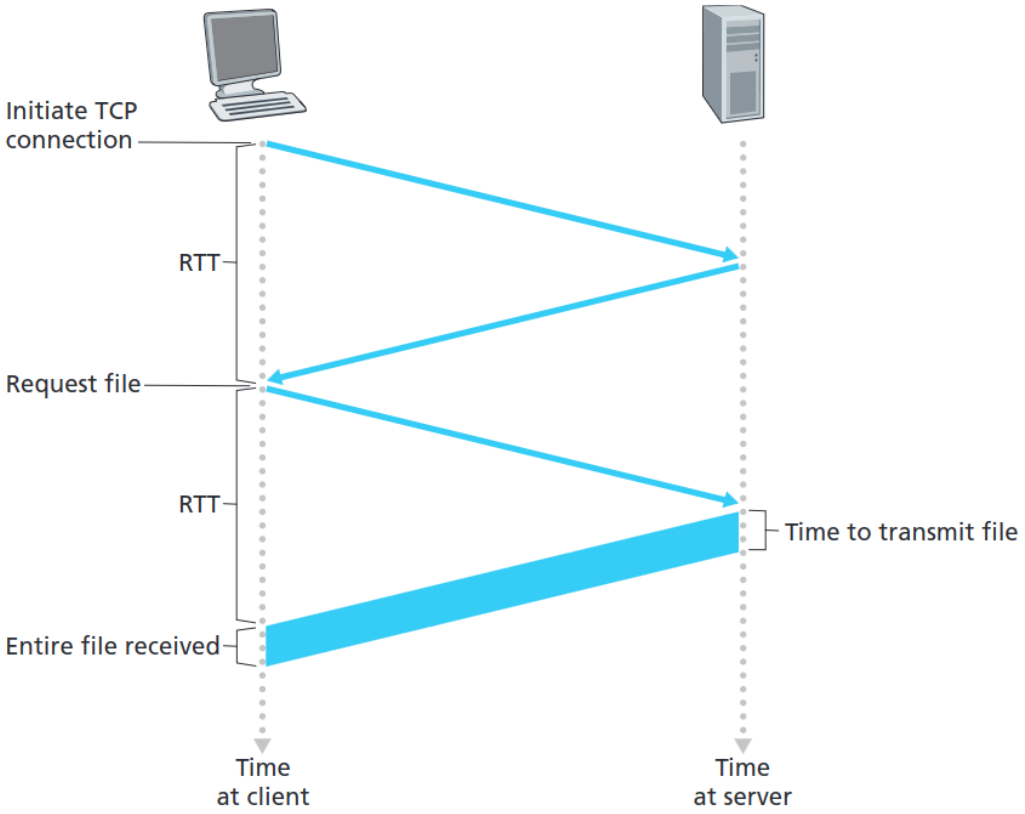


Fig. 13: Sessão TCP não persistente

Suponhamos o começo de uma sessão TCP entre um *browser* e um *Web server*. É inicializado o procedimento *three-way handshake*: O cliente (*browser*) envia um pequeno segmento TCP ao servidor, subsequentemente o servidor recebe-o com sucesso e envia de volta um outro segmento TCP. Após completar as primeiras duas partes do *handshake* o cliente envia um pedido HTTP incluindo também a terceira parte do *handshake* (o reconhecimento da receção do segundo segmento). A conexão é solidificada, falta agora atender ao pedido:

1. O servidor HTTP busca o objeto de pesquisa, encapsula-o numa mensagem de resposta HTTP e envia-a para o cliente através da *socket*.
2. O processo servidor indica o termínio da sessão TCP (embora esta não feche até saber o cliente recebeu a mensagem de forma intact).
3. O cliente recebe a mensagem e consequentemente a sessão TCP é terminada.

O tempo total de resposta é:

$$\boxed{\text{RTT} + \text{RTT} + \text{Time to transmit file}}$$

Onde RTT (*round-trip time*) é o tempo que um segmento TCP demora a viajar do cliente ao servidor e viceversa. "The RTT includes packet-propagation delays, packet-queuing delays in intermediate routers and switches, and packet-processing delays."[1]

Nota: As sessões TCP não persistentes podem decorrer em paralelo, "most browsers open 5 to 10 parallel TCP connections, and each of these connections handles one request-response transaction (*forking*)"

⋆ Sessão persistente

Na sessões TCP persistentes ocorrem múltiplas transações HTTP por sessão TCP e o servidor não fecha a sessão TCP depois de responder a um pedido. Ocorre canalização (*pipelining*) de pedidos, as respostas são enviadas na ordem de receção:

"Non-persistent connections in HTTP have limitations, such as the need to establish a new connection for each requested object and a two RTT delivery delay for each object. HTTP 1.1 introduced persistent connections, which allow the server to keep the TCP connection open for multiple requests and responses between the client and server. This enables sending an entire web page or multiple pages from the same server over a single persistent TCP connection. **Requests can be made back-to-back (pipelining) without waiting for replies to pending requests, and the server typically closes the connection after a configurable timeout interval of inactivity**."

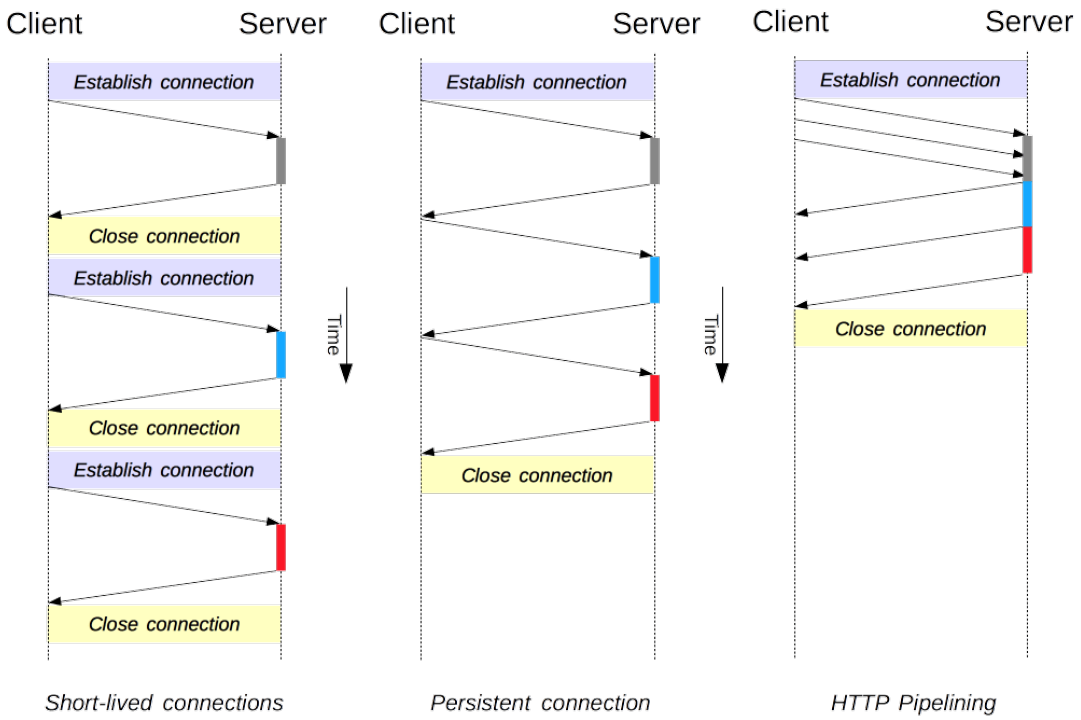


Fig. 14: Connection management in HTTP.

→ Transfer-Encoding: chunked

Chunked encoding is a method used in HTTP to transmit data in smaller, manageable pieces (*chunks*) instead of sending the entire resource in one go.

1. **Purpose:** Enable the server to start transmitting data without knowing the total size of the resource, which is useful for dynamic content generation or when the resource size is unknown in advance.
2. **Chunks:** Data is divided into separate chunks, each with its own size specified at the beginning. The size is followed by the actual data and a newline character.
3. **Transmission:** The server sends each chunk one by one, with the client assembling the received chunks to reconstruct the original resource.
4. **End of data:** A special chunk with a size of zero signals the end of the data transmission, allowing the client to know when it has received the entire resource.
5. **Benefits:** Improved responsiveness, as the client can start processing data as soon as the first chunk is received, and reduced resource usage, as the server doesn't need to buffer the entire resource before sending.

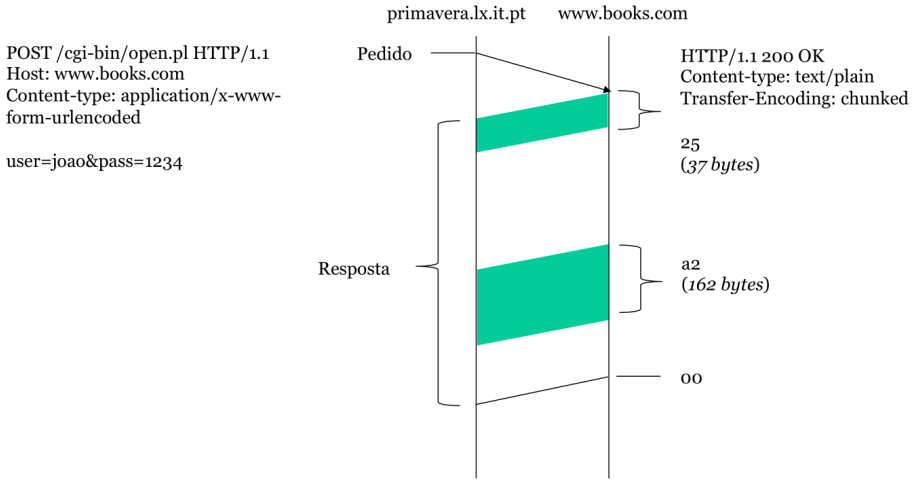


Fig. 15: "Transfer-Encoding: chunked"[3]

→ Redirection

Redirection is a mechanism used in HTTP to inform a client that a requested resource is available at a different URL and instruct it to retrieve the resource from that new location.

1. **Purpose**: Enable web servers to move or rename resources, load balance requests across multiple servers, or enforce URL canonicalization for SEO purposes.
2. **HTTP status codes**: Redirection is signaled using HTTP status codes in the 3xx range, with each code indicating a specific type of redirection (e.g., 301 Moved Permanently, 302 Found, 303 See Other, 307 Temporary Redirect).
3. **Location header**: The server provides the new URL for the resource in the Location HTTP header of the response. The client should follow the new URL to access the requested resource.
4. **Considerations**: Excessive or circular redirections can lead to poor user experience and increased server load. It is essential to use the appropriate status codes and manage redirects properly to avoid such issues.

→ Formato de mensagens HTTP

HTTP communication relies on a request-response model, where the client initiates a request, and the server sends back a response. This exchange typically consists of four components:

1. **Request/Response Line**
2. **Headers:** Provides metadata about the request or response, such as content type, content length, and encoding.
3. **Blank Line:** A separator between the headers and the message body.
4. **Message Body (optional):** Contains the data being sent in the request or response, such as an HTML document or form data.

⋆ HTTP request

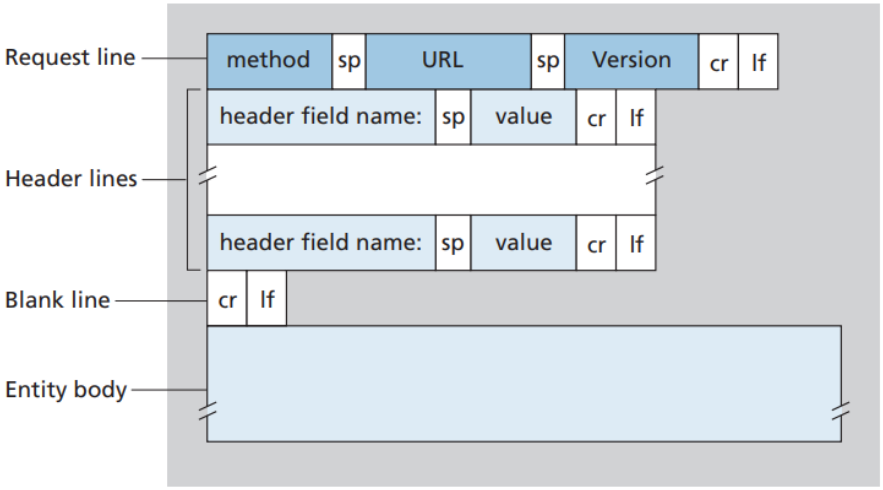


Fig. 16: "General format of an HTTP request message."[1]

HTTP defines several methods (also known as verbs) that indicate the action to be performed on the requested resource. Some of the most common methods include:

- `GET`: Requests data from a specified resource.
- `POST`: Submits data to a specified resource for processing, usually resulting in a change on the server.
- `PUT`: Updates the specified resource with the supplied data.
- `DELETE`: Deletes the specified resource.
- `HEAD`: Similar to `GET`, but only requests the headers, without the actual data.
- `OPTIONS`: Retrieves the communication options available for the specified resource.

[§]A *Uniform Resource Locator (URL)* is a standardized address format used to locate resources on the internet. URLs typically consist of a protocol (e.g., `http`, `https`), a domain name or IP address, an optional port number, and a path to the specific resource. For example, `https://example.com:80/path/to/resource` is a URL, where `https` is the protocol, `example.com` is the domain name, `80` is the port number, and `/path/to/resource` is the path to the desired resource.

⋆ HTTP response

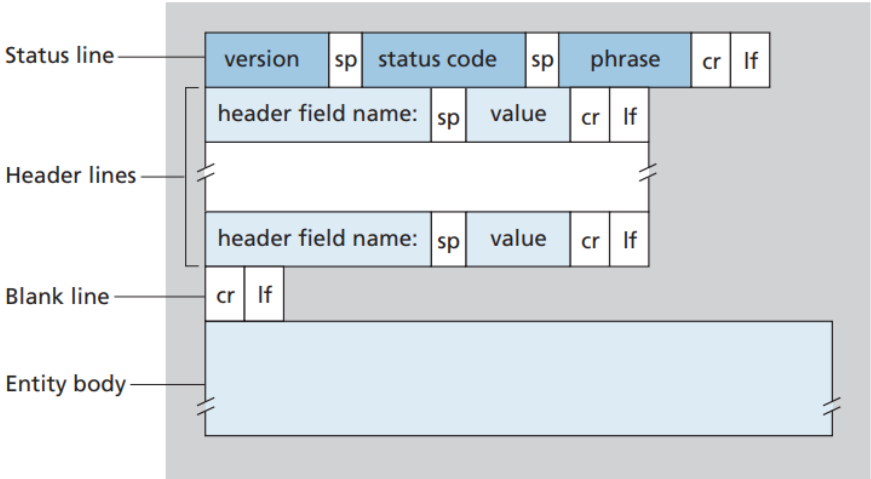


Fig. 17: "General format of an HTTP response message."[1]

HTTP status codes are three-digit numbers included in server responses that indicate the result of the client's request. They are categorized into five classes, based on the first digit:

- **1xx (Informational):** Request received, and the server is continuing to process it.
- **2xx (Successful):** Request successfully received, understood, and accepted.
- **3xx (Redirection):** Request needs further action to be completed, such as following a different URL.
- **4xx (Client Error):** The request contains bad syntax or cannot be fulfilled by the server.
- **5xx (Server Error):** The server failed to fulfill a valid request.

→ Interação Cliente-Servidor: *Cookies*

Cookies are small text files stored on a user's device, allowing web applications to remember stateful information across sessions, such as preferences or login data.

1. **Creation:** Servers generate cookies and send them to user's devices via `Set-Cookie` HTTP headers. The user's web browser then stores the cookie on the device.
2. **Cookie usage:** In subsequent requests to the same website, the user's web browser automatically sends the cookie back to the server via the Cookie HTTP header. The server can then use the information in the cookie to customize the user's experience, for example, by displaying a message or remembering items in a shopping cart.
3. **Cookie Management:** Websites can update or delete cookies; users can manage them via browser settings.
4. **Privacy concerns:** While cookies provide a convenient way to maintain state across multiple sessions, they can also raise privacy concerns due to their ability to track users' browsing habits and collect personal information. Regulations such as the EU's General Data Protection Regulation (GDPR) aim to protect user privacy by requiring websites to obtain user consent before setting cookies and providing options for users to opt-out of tracking.

→ *Web caching*

Web cachingis a technique to improve the efficiency and reduce the latency of web access by storing copies of frequently requested resources on a local cache server—also called a **proxy server**.

1. **Purpose:** Reduce the load on origin servers, decrease network traffic, and improve user experience by providing faster response times.
2. **Cache server:** A dedicated server that stores copies of resources, such as images or HTML files, to serve user requests without contacting the origin server.
3. **Cache hit:** When a requested resource is found in the cache server, it's called a cache hit. The cache server then serves the resource directly to the user.
4. **Cache miss:** When a requested resource is not found in the cache server, it's called a cache miss. The cache server must fetch the resource from the origin server, store it locally, and serve it to the user.
5. **Cache replacement:** Due to limited storage capacity, cache servers employ replacement policies (e.g., Least Recently Used) to determine which resources to evict when the cache is full.
6. **Conditional `GET`:** To ensure content freshness, cache servers can issue conditional `GET` requests, which only retrieve a resource if it has changed since the cache server's last fetch.
7. **Collaborative caching:** Cache servers can work together to share resources, reducing redundant downloads and improving overall caching efficiency.

⋆ Conditional `GET`

1. `If-Modified-Since`**:** The client includes the `If-Modified-Since` HTTP header in its request, specifying the date and time of its last request for the resource. The server checks whether the resource has been modified since the specified date and time.
2. **Resource status:** If the resource has been modified, the server sends the updated resource along with a 200 OK status code. If the resource has not been modified, the server sends a 304 Not Modified status code without the resource content, indicating that the client's cached version is still valid.
3. `ETag` **and** `If-None-Match`**:** An alternative method uses the `ETag` (Entity Tag) header, which represents a unique identifier for the resource's current version. The client includes the `If-None-Match` header in its request, containing the `ETag` value received in the previous response. If the server's current `ETag` for the resource does not match the value in the `If-None-Match` header, the server sends the updated resource and its new `ETag`. If the `ETags` match, the server responds with a 304 Not Modified status code.
4. **Cache validation:** Conditional `GET` is a crucial component of HTTP caching, as it allows clients to validate their cached resources and refresh them when necessary, without having to download the entire resource each time.
5. **Performance benefits:** By reducing the amount of data transferred and allowing clients to reuse cached resources, conditional `GET` improves network efficiency, reduces server load, and results in faster load times for end users.

→ HTTP/2: Bloqueio topo-da-fila (*head-of-line blocking*)

Head-of-line blocking (HOL)

"Head-of-line blocking (HOL blocking) in computer networking is a performance-limiting phenomenon that occurs when a line of packets is held up in a queue by a first packet."

A versão HTTP/1.1 procura evitar este fenómeno recorrendo à abertura de várias sessões TCP paralelas, permitindo a renderização de objetos mais pequenos e subsequentemente diminuindo o *user-perceived delay*. No entanto, o mecanismo de controlo de congestão inerente ao protocolo TCP força a abertura inadvertida de múltiplas conexões paralelas (exaustando assim as *sockets* do sistema operativo):

> "TCP congestion control inadvertently encourages browsers to use multiple parallel TCP connections instead of a single persistent connection. This is because congestion control aims to distribute an equal share of the available bandwidth to each TCP connection sharing a bottleneck link. By opening multiple parallel connections, a browser can "cheat" and acquire a larger portion of the link bandwidth."[1]

A versão HTTP/2 procura otimizar o uso da funcionalidade de controlo de congestão bem como reduzir o número de conexões paralelas abertas:

- **Framing e Multiplexagem de streams**: HTTP/2 breaks messages into small frames and interleaves request and response messages on the same TCP connection, significantly reducing user-perceived delay. The framing sub-layer of HTTP/2 breaks down HTTP messages into independent frames and reassembles them on the other end. This allows for interleaving and more efficient processing.
- **Binary encoding e compressão**: Frames are binary encoded (uma frame para o cabeçalho e outras múltiplas para o corpo da resposta), making them more efficient to parse, slightly smaller, and less error-prone.
- **Response message prioritization**: Clients can prioritize responses by assigning a weight between 1 and 256, allowing the server to send higher-priority frames first. Clients can also state each message's dependency on other messages.
- **Server pushing**: HTTP/2 enables servers to send multiple responses for a single client request, pushing additional objects to the client without the client having to request each one. This eliminates extra latency due to waiting for requests. (Servidor envia objetos que sabe que o cliente vai precisar sem que este lhe envie os pedidos correspondentes).

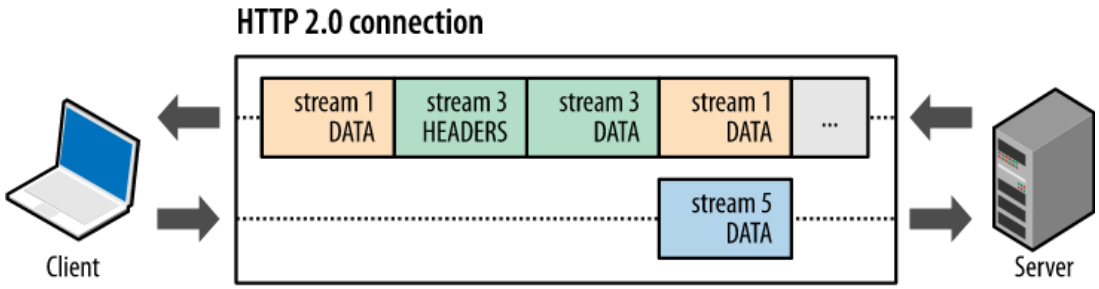


Fig. 18: Multiplexagem de *streams* de *frames*

↳ Eletronic Mail (email)

→ Simple Mail Transfer Protocol (SMTP)

Simple Mail Transfer Protocol (SMTP)

"SMTP is the principal application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As with most application-layer protocols, SMTP has two sides: a client side, which executes on the sender's mail server, and a server side, which executes on the recipient's mail server."[1]

Nota: "SMTP restricts the body (not just the headers) of all mail messages to simple 7-bit ASCII" & "does not normally use intermediate mail servers for sending mail, even when the two mail servers are located at opposite ends of the world."´. Assim, se o servidor de destino estiver inativo, a receção da mensagem está dependente do reenvio por parte do servidor do utilizador, não permanecendo num servidor de resguardo temporário.

Exemplo SMTP [3]

```
S: 220 destino.pt
C: HELO origem.pt
S: 250 Hello origem.pt, pleased to meet you
C: MAIL FROM: fernando@origem.pt
S: 250 fernando@origem.pt ... Sender ok
C: RCPT TO: luis@destino.pt
S: 250 luis@destino.pt ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: From: fernando@origem.pt
C: To: luis@destino.pt
C:
C: Deus quer, o homem sonha, a obra nasce
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 destino.pt closing connection
```

- The client sends a message ("Deus quer, o homem sonha, a obra nasce") from mail server `origem.pt` to mail server `destino.pt`.
- The client issued five commands: `HELO` (an abbreviation for HELLO), `MAIL FROM`, `RCPT TO`, `DATA`, and `QUIT`.
- The client also sends a line consisting of a single period, which indicates the end of the message to the server.
- The server issues replies to each command, with each reply having a reply code and some (optional) English-language explanation (e.g.: `354 Enter mail, end with "." on a line by itself`).
- SMTP uses persistent connections: If the sending mail server has several messages to send to the same receiving mail server, it can send all of the messages over the same TCP connection.
- For each message, the client begins the process with a new `MAIL FROM: origem.pt`, designates the end of the message with an isolated period, and issues `QUIT` only after all messages have been sent.

→ Formato da mensagem

O formato típico atribuído ao cabeçalho de um email é:

Exemplo do formato do cabeçalho de uma mensagem

```
From: fernando@origem.pt
To: luis@destino.edu
Subject: Tenho em mim todos os sonhos do mundo
```

O cabeçalho é seguido de uma linha em branco e do corpo do email.

Nota: "It is important to note that these header lines are different from the SMTP commands (even though they contain some common words such as "from" and "to"). The commands in that section were part of the SMTP handshaking protocol; the header lines examined in this section are part of the mail message itself."[1]

→ Mail access protocol

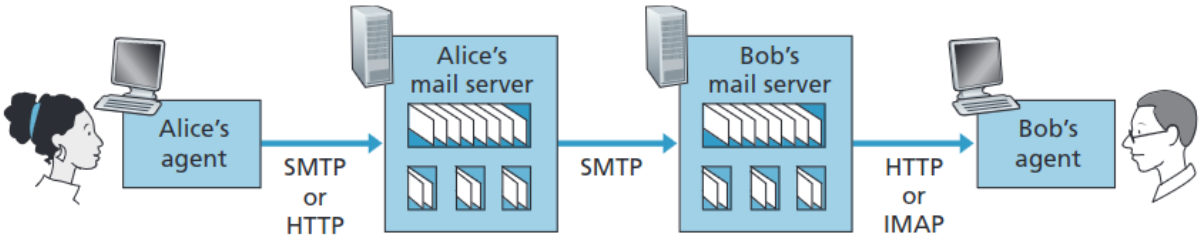


Fig. 19: Protocolos de *push* e *pull*

- Alice's user agent uses SMTP or HTTP to deliver the e-mail message into her mail server, then Alice's mail server uses SMTP (as an SMTP client) to relay the e-mail message to Bob's mail server. The two step procedure allows Alice's mail server to repeatedly try to send the message to Bob's mail server, until Bob's mail server becomes operational.
- On the other hand, if Bob is using Web-based e-mail or a smartphone app (such as Gmail), then the user agent will use HTTP (since obtaining the messages is a pull operation, whereas SMTP is a push protocol) to retrieve Bob's e-mail. This case requires Bob's mail server to have an HTTP interface as well as an SMTP interface (to communicate with Alice's mail server).

The alternative method, typically used with mail clients such as Microsoft Outlook, is to use the Internet Mail Access Protocol (IMAP). Both the HTTP and IMAP approaches allow Bob to manage folders, maintained in Bob's mail server.

[§]**Internet Message Access Protocol (IMAP):** managing and accessing email messages on a remote.

1. **Server-based Storage:** IMAP stores email messages on a remote server, allowing users to access their emails from multiple devices without needing to download messages locally.
2. **Folder Organization:** Users can create, rename, and delete folders to organize their emails.
3. **Selective Retrieval:** IMAP supports downloading only email headers or specific parts of a message, conserving bandwidth and improving efficiency.
4. **Synchronization:** Changes made to emails, such as marking messages as read or moving them between folders, are synchronized with the server, ensuring a consistent view across all devices.

↳ Domain Name System (DNS)

The Domain Name System (DNS) is a hierarchical and distributed database that translates human-readable domain names (e.g., `www.example.com`) into their corresponding IP addresses, which are used by computers to communicate with each other. This translation is essential for the functioning of the Internet.

→ Services Provided by DNS

DNS provides various services, including:

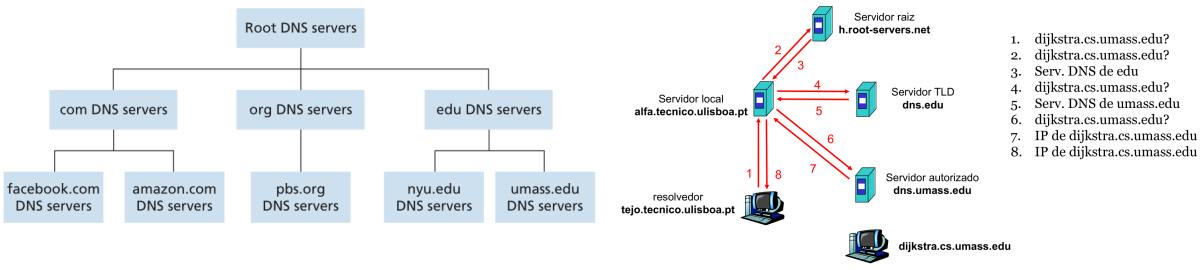
1. **Hostname-to-IP address translation:** The primary function of DNS is to translate domain names into their corresponding IP addresses.
2. **Alias resolution:** DNS allows the use of aliases (alternative domain names) to reference the same IP address. This is helpful for load balancing and providing user-friendly names.
3. **Reverse lookup:** DNS can perform a reverse lookup, converting an IP address back into its corresponding domain name.
4. **Load distribution:** DNS can distribute the load among multiple servers by returning different IP addresses for the same domain name.

→ Overview of how DNS works

DNS operates using a distributed hierarchy of servers organized into four levels:

1. **Root DNS servers:** These servers are at the top of the hierarchy and provide information about top-level domains (e.g., .com, .org).
2. **Top-level domain (TLD) servers:** These servers are responsible for specific top-level domains, such as .com or .org, and provide information about second-level domains (e.g., example.com).
3. **Authoritative DNS servers:** These servers store the actual DNS records for specific domain names and are responsible for providing the IP address associated with a given domain name.
4. **Local DNS servers:** These servers are operated by ISPs or organizations and are the first point of contact for end-user DNS queries. They cache query results to provide faster responses for future requests.

When a user requests the IP address for a domain name, the local DNS server initiates a series of queries through the DNS hierarchy until it reaches the authoritative DNS server for the requested domain, which provides the IP address.



(a) "Portion of the hierarchy of DNS servers"[1] (b) "DNS: exemplo"[3]

Fig. 20: DNS hierarchy overview

→ DNS records and messages

DNS uses **Resource Records** (RRs) to store information. The most common RR types are:

1. **A:** Contains the IP address corresponding to a given domain name.
2. **NS:** Specifies the authoritative name server for a domain.
3. **CNAME:** Defines an alias for a domain name, allowing multiple names to reference the same IP address.
4. **MX:** Indicates the mail exchange server responsible for handling email for a domain.
5. **PTR:** Provides reverse lookup functionality, mapping an IP address back to a domain name.

Exemplo de registo de recursos [3]

```
(Domain name, Value, Type, Classe, TTL)
```

```
(tejo.tecnico.ulisboa.pt, 193.136.138.142, A, IN, 448)
(tejo.tecnico.ulisboa.pt, 2001:CD00:0:CDE:1257:0:211E:729C, AAAA, IN, 448)
(tecnico.ulisboa.pt, ns1.tecnico.ulisboa.pt, NS, IN, 3600)
(mae.princeton.edu, live-princeton-mae-next.pantheonsite.io, CNAME, IN, 86400)
(tecnico.ulisboa.pt, email.tecnico.ulisboa.pt, MX, IN, 3600)
```

DNS messages consist of two types of messages:

1. **Query:** A message sent by a client (local DNS server) to a DNS server, requesting the resolution of a domain name or an IP address.
2. **Response:** A message sent by the DNS server back to the client, containing the requested information (e.g., IP address or domain name) and any additional relevant records.

Both query and response messages have the same format, consisting of a header and four sections: Question, Answer, Authority, and Additional Information. The header contains various flags and identifiers, while the four sections contain the relevant resource records and requested information.

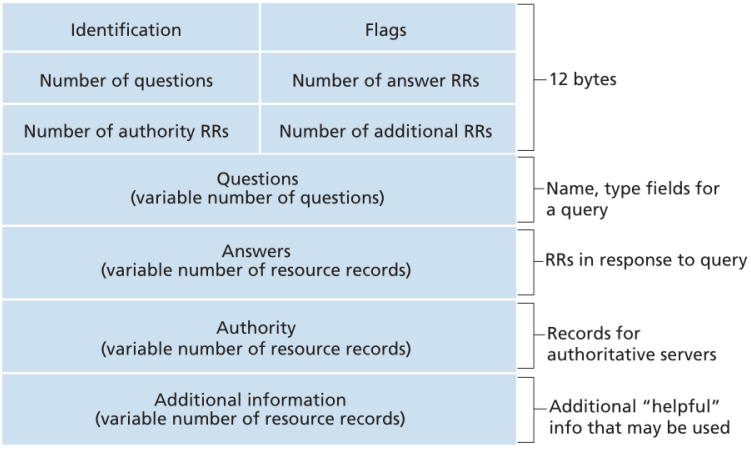


Fig. 21: "DNS message format"[1]

§"`TTL` is the Time-To-Live of the resource record; it determines when a resource should be removed from a cache."[1]

→ DNS Caching and Query Resolution

1. **DNS Caching:** Temporarily storing DNS query results to reduce lookup times for subsequent requests.
   (a) **Local Cache:** DNS clients (resolvers) and recursive DNS servers store query results locally to quickly answer repeated requests.
   (b) **Cache Expiration:** Each DNS record has a Time-To-Live (TTL) value determining how long it should be stored in a cache before being discarded.
2. **Recursive Query:** A DNS server contacts other DNS servers on behalf of the client to resolve a query, returning the final result to the client.
3. **Iterative Query:** A DNS server provides the client with a referral to another DNS server, and the client continues the query resolution process by contacting the referred server.

↳ Socket Programming: Aplicações de Rede

Uma aplicação de rede é tipicamente constituída por um par de programas—um programa de *cliente* e um programa de *servidor*—que residem em sistemas terminais distintos. Quando estes programas são executados, é iniciado um *processo de cliente* e um *processo de servidor*, que comunicam entre si ao ler e escrever em ***sockets***.

Socket

"A **network socket** is a software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network. The structure and properties of a socket are defined by an application programming interface (API) for the networking architecture. Sockets are created only during the lifetime of a process of an application running in the node. The API for the network protocol stack creates a handle for each socket created by an application, commonly referred to as a *socket descriptor*."

As *sockets* são uma abstração que permite comunicar entre sistemas terminais com o uso de *socket descriptors* standard. Em sistemas UNIX, todas as ações `I/O` são realizadas ao escrever, e ler, em *file descriptors*.

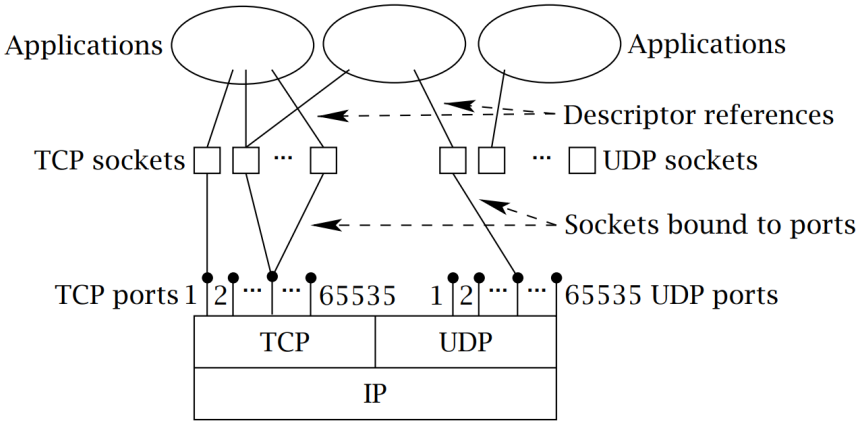


Fig. 22: "Logical relationships among applications, socket abstractions, protocols, and port numbers within a single host."[2]

[§]**Nota:** Um *file descriptor* (`fd`) é um inteiro associado a um ficheiro aberto que pode ser: uma conexão de rede, um ficheiro de texto, um terminal...

3. Camada de Transporte

↳ Visão Geral

"Recall that the transport layer lies just above the network layer in the protocol stack. Whereas a **transport-layer protocol provides logical communication between processes** running on different hosts, a **network layer protocol provides logical communication between hosts**."[1]

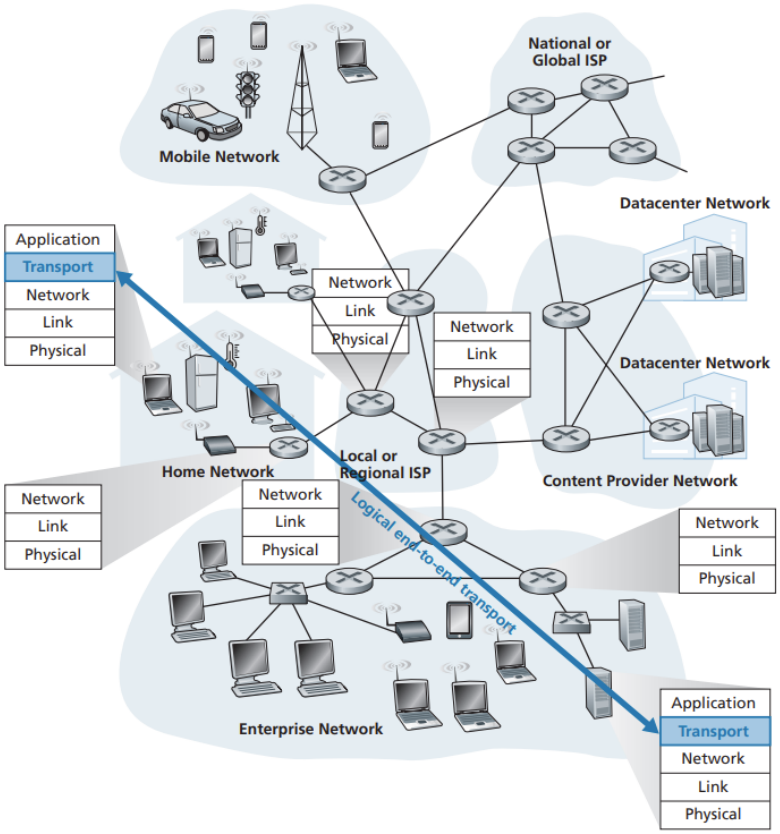


Fig. 23: "Logical communication between processes"[1]

A camada de transporte fornece uma abstração sobre a comunicação entre aplicações (processos) que atuam em *hosts* diferentes—"It allows for communication between processes running on different hosts as if they were directly connected, even if they are physically located on opposite sides of the planet"[1]. A camada de rede garante o fluxo de *packets* de um *host* a outro, iterando sobre um número arbitrário de redes.

"Recall that the Internet makes two distinct transport-layer protocols available to the application layer. One of these protocols is **UDP** (User Datagram Protocol) and **TCP** (Transmission Control Protocol)":

- **Entrega fiável e ordenada para unicast: TCP**
  - Controlo de erros
  - Controlo de fluxo
  - Controlo de congestão
- **Entrega não-fiável para unicast ou multicast: UDP**
  - Deteção de erros

> "Transport-layer protocols are implemented in the end systems but not in network routers. On the sending side, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer **segments** in Internet terminology. This is done by (possibly) breaking the application messages into **smaller chunks** and adding a **transport-layer header to each chunk** to create the **transport-layer segment**. The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a datagram) and sent to the destination."[1]

↳ Multiplexação e Desmultiplexação

Multiplexing and demultiplexing

"At the destination host, the transport layer receives segments from the network layer just below. The transport layer has the responsibility of delivering the data in these segments to the appropriate application process (socket) running in the host. (...) **This job of delivering the data in a transport-layer segment to the correct socket is called demultiplexing**. **The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information to create segments**, and passing the segments to the network layer **is called multiplexing**"

A multiplexação e a desmultiplexação está dependente da identificação única inerente às *sockets* e requer que cada segmento da camada de transporte possua um **source port number** field e um **destination port number** field (*vide* Appendix C: Reserved Ports).

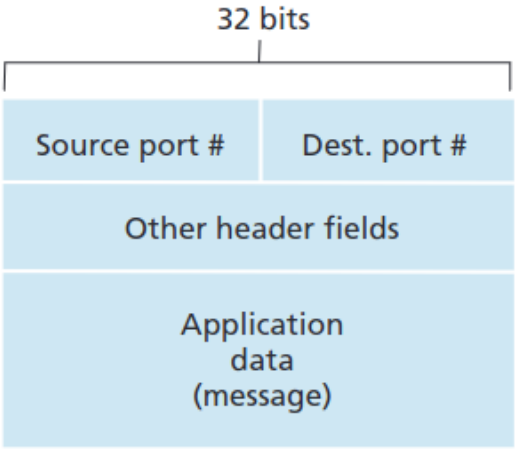


Fig. 24: Estrutura de um segmento da camada de transporte.[1]

Os restantes *header fields* estão dependentes do tipo de protocolo em uso, nomeadamente, **UDP** e **TCP**.

↳ Connectionless Transport: UDP

Nota: As sockets UDP são identificada somente por dois campos: porto de destino e endereço IP de destino. Tal significa que informação oriunda de dois *hosts* diferentes (consequentemente IP's de origem diferentes) cujos destinos possuem o mesmo IP e o mesmo porto será recebida pela mesma *socket*.

A estrutura de um segmento UDP é a seguinte:

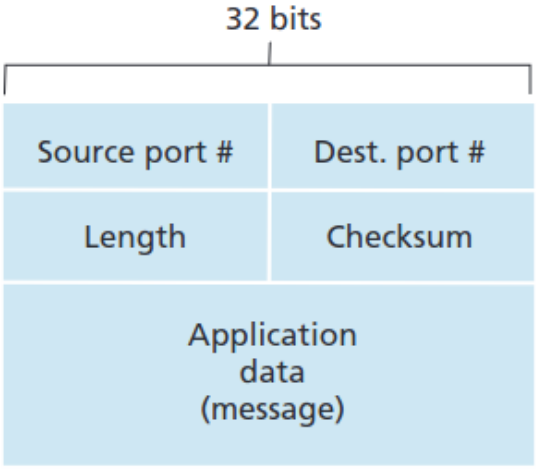


Fig. 25: Estrutura de um segmento UDP

- The UDP header has only four fields, each consisting of two bytes.
- The port numbers allow the destination host to pass the application data to the correct process running on the destination end system (that is, to perform the demultiplexing function).
- The length field specifies the number of bytes in the UDP segment (header plus data). An explicit length value is needed since the size of the data field may differ from one UDP segment to the next.
- The checksum is used by the receiving host to check whether errors have been introduced into the segment (como já referido, o protocolo garante apenas deteção de erros).

→ Checksum

Checksum (soma de verificação): $\overline{x_1 \oplus \cdots \oplus x_m} = 2^n - 1 - (x_1 \oplus \cdots \oplus x_m)$

"[The] Sender side performs the 1s complement of the sum of all the 16-bit words in the segment"[1]

"The UDP checksum provides for error detection. That is, **the checksum is used to determine whether bits within the UDP segment have been altered** (for example, by noise in the links or while stored in a router) as it moved from source to destination."[1]

Relembrar: Soma binária com *wrapping* em caso de *overflow*

$$x \oplus y = \begin{cases} x + y, & x + y < 2^n \\ x + y - 2^n + 1, & x + y \geqslant 2^n \end{cases}$$

Exemplo: Supondo um segmento composto por 3 palavras de 16 bits.

```
0110011001100000
0101010101010101
1000111100001100
```

The sum of first two words is:

```
0110011001100000
0101010101010101
----------------
1011101110110101
```

→

Adding the third word gives:

```
1011101110110101
1000111100001100
----------------
0100101011000010
```

∴ The 1s complement of the sum `0100101011000010` is `1011010100111101`

Lastly, "At the **receiver**, all four 16-bit words are added [the segment's words and the checksum]. If no errors are introduced into the packet, then clearly the sum at the receiver will be `1111111111111111`."[1]

Nota: Neste exemplo, a ultima soma resulta num *overflow* que sofre um *wrap around*, i.e., o bit de excesso é somado ao bit menos significativo do resultado.

↳ Reliable Data Transfer

→ Stop-and-Wait

Stop-and-wait is a simple and fundamental technique for reliable data transfer. The sender transmits a single data packet and then waits for an acknowledgment (ACK) from the receiver before transmitting the next packet.

1. **Packet numbering**: To detect and discard duplicate packets, both data packets and ACKs are numbered. The receiver can then distinguish between new packets and duplicates.
2. **Error detection**: Both data packets and ACKs include a checksum for error detection. If the receiver detects an error in a packet, it discards the packet without sending an ACK, prompting the sender to retransmit the packet.
3. **Timeout and retransmission**: The sender sets a timer after transmitting a packet. If the timer expires before receiving an ACK, the sender assumes the packet or the ACK was lost and retransmits the packet.

   "If we define the utilization of the sender (or the channel) as the fraction of time the sender is actually busy sending bits into the channel, [direct observation] (...) shows that the stop-and-wait protocol has a rather **dismal sender utilization** (...)"[1]

$$U_{\text{sender}} = \frac{L/R}{\text{RTT} + L/R} \ll 1$$

The solution to this *conundrum* is known as **pipelining** as discussed bellow, where two basic approaches toward pipelined error recovery can be identified.

→ Cumulative ACKs and Selective ACKs

1. **Cumulative ACKs**: In this approach, the receiver acknowledges the receipt of all consecutive, correctly received packets up to a specified sequence number by sending an ACK containing the sequence number of the next expected packet. If a packet is lost or arrives out of order, the receiver sends an ACK with the sequence number of the first expected packet it has not received.
2. **Selective ACKs**: With selective ACKs, the receiver explicitly acknowledges individual packets, allowing the sender to retransmit only the missing packets. This approach improves efficiency, especially in scenarios with high packet loss or long round-trip times.

→ Sliding Window Protocol

The sliding window protocol is a general concept that forms the basis for both Go-Back-N (GBN) and Selective Repeat (SR) protocols. It allows the sender to transmit multiple packets without waiting for an acknowledgment (ACK) for each one. The sender maintains a sending window that limits the number of unacknowledged packets, while the receiver maintains a receiving window that limits the number of out-of-order packets it can accept. The windows slide as the sender receives ACKs and the receiver receives in-order packets.

Relação temporal fundamental: $\boxed{\frac{L}{c} \ll \text{RTT} < W \cdot \frac{L}{c} < Timeout}$

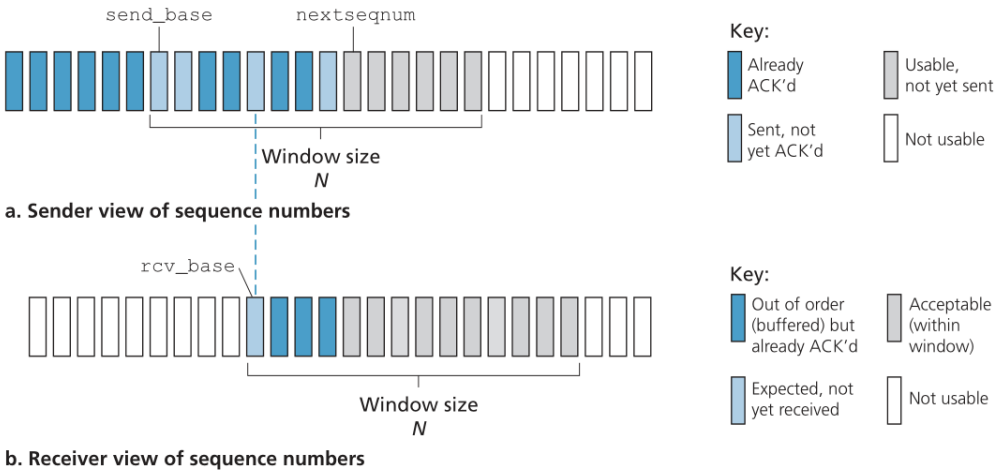


Fig. 26: Exemplo janela deslizante

⋆ Go-Back-N (GBN)

Go-Back-N (GBN) is a specific implementation of the sliding window protocol where the receiver sliding window has a size of 1. It uses a cumulative acknowledgment scheme. If a packet is lost or corrupted, the receiver discards all out-of-order packets received after the lost packet. When the sender detects the loss, it retransmits the lost packet and all subsequent packets. GBN is relatively simple to implement, but its performance degrades in networks with high error rates, as it requires retransmission of multiple packets even if only a single packet was lost.

Tamanho máximo da janela para evitar ambiguidades na receção: $\boxed{W_s \leqslant N_{\text{seq}} - 1}$

⋆ Temporizador/janela e ACKs cumulativos

Uma alternativa ao protocolo GBN é a utilização de uma janela de tamanho igual à sender window no lado do receiver. Deste modo, a retransmissão de todos os pacotes subsequentes é evitada.

Tamanho máximo da janela para evitar ambiguidades na receção: $\boxed{W \leqslant \lfloor N_{\text{seq}}/2 \rfloor}$

⋆ Selective Repeat (SR). Temporizador/pacote e ACKs seletivos

Selective Repeat (SR) is another implementation of the sliding window protocol, which addresses the performance issues of GBN. Instead of using cumulative ACKs, the receiver sends individual ACKs (selective ACKs) for each received packet, regardless of their order. The receiver stores out-of-order packets in a buffer until the missing packets arrive. The sender only retransmits the lost or corrupted packets, which are identified by the absence of their respective ACKs. Although SR is more efficient than GBN in handling packet losses, it is more complex to implement due to the need to handle individual ACKs and maintain the buffer for out-of-order packets.

Tamanho máximo da janela para evitar ambiguidades na receção: $\boxed{W \leqslant \lfloor N_{\text{seq}}/2 \rfloor}$

→ Fast Retransmission with 3 ACKs

Fast retransmission is an optimization technique to improve the performance of reliable data transfer protocols. It allows the sender to detect and retransmit lost packets faster than relying solely on timeouts.

1. **Duplicate ACKs**: When the receiver detects an out-of-order packet, it sends a duplicate ACK sends an ACK with the sequence number of the first expected packet it has not received. The sender can infer that a packet was lost if it receives multiple duplicate ACKs.
2. **Three-duplicate-ACKs rule**: If the sender receives three duplicate ACKs for the same packet, it assumes the packet was lost and immediately retransmits it without waiting for the timeout.

→ Reminder

⋆ Canal não sequencial

Mediante um canal não fiável não sequencial, o espaço de numeração deve ser tal que não existam repetições de identificadores dentro do intervalo de tempo decorrido entre a recepção do pacote mais rápido e a recepção do pacote mais lento, sendo o pacote mais lento emitido imediatamente antes do mais rápido:

$$\boxed{\text{número de identificadores} \geqslant r \cdot T}$$

Onde r é o débito a que os números de sequência são consumidos (1/delay de transmissão para *sliding window protocol* e 1/RTT para *stop-and-wait protocol*) e T é o tempo de vida máxima de um pacote no canal.

⋆ Transmissão sem interrupção

$$\boxed{N_{\text{pkt}}\, t_{\text{trans}} \geqslant \text{RTT} + t_{\text{trans}}}$$

→ TL;DR Reliable Data Transfer

Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Window, pipelining	The sender may be restricted to sending only packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. We'll see shortly that the window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both.

Tab. 3: "Summary of reliable data transfer mechanisms and their use"[1]

↳ Connection-Oriented Transport: TCP

→ The TCP Connection

- **Connection-oriented:** TCP establishes a connection through a handshake before data can be exchanged between application processes.
- **Logical connection:** The TCP connection is not an end-to-end circuit but a logical one, with common state residing only in the TCPs of the communicating end systems.
- **Full-duplex service**: TCP connections allow simultaneous data flow in both directions between two processes.
- **Point-to-point:** TCP connections only involve a single sender and a single receiver, with multicasting not possible.
- **Three-way handshake:** To establish a TCP connection, the client and server exchange three special TCP segments, setting up the parameters for the data transfer.
- **Send and receive buffers:** Each side of the connection has its own send buffer and its own receive buffer. The client process passes a stream of data through its socket, and TCP directs this data to the connection's send buffer. From time to time, TCP grabs chunks of data from the send buffer and passes it to the network layer.
- **Maximum Segment Size (MSS):** Maximum amount of application-layer data in a segment, typically determined by the **Maximum Transmission Unit (MTU)** for the local sending host. "Ethernet and PPP link-layer protocols have an MTU of 1,500 bytes. Thus, a typical value of MSS is 1460 bytes."[1]
- **TCP segments:** TCP pairs each chunk of client data with a TCP header, forming TCP segments, which are passed to the network layer and encapsulated within network-layer IP datagrams.

→ TCP Segment Structure

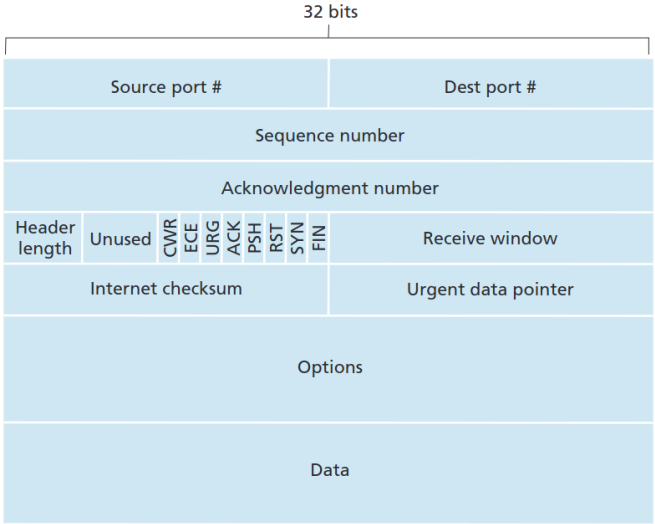


Fig. 27: Estrutura de um segmento TCP (consists of a header and payload). The header contains essential information, such as source and destination port numbers, sequence numbers, acknowledgment numbers, and flags. Additionally, the header includes the window size, checksum, and urgent pointer fields.

- **Sequence Numbers and Acknowledgment Numbers**
  - Sequence numbers are over the stream of transmitted bytes, not over the series of transmitted segments.
  - Acknowledgment numbers indicate the sequence number of the next byte the host is expecting.
  - TCP provides **cumulative acknowledgments**, only acknowledging bytes up to the first missing byte in the stream.
  - Initial sequence numbers are randomly chosen to minimize possible confusion with segments from earlier connections.
- **Header Fields**
  - **Source and destination port numbers**: for multiplexing/demultiplexing data from/to upper-layer applications.
  - **Checksum field**
  - 32-bit **sequence number field** and 32-bit **acknowledgment number field**: used for reliable data transfer service
  - 16-bit **receive window field**: used for flow control, indicating the number of bytes that a receiver is willing to accept
  - 4-bit **header length field**: specifies the length of the TCP header in 32-bit words
  - **Optional** and **variable-length options field**: used for negotiating the MSS, window scaling factor, and time-stamping option
  - **Flag field**: contains 6 bits (ACK, RST, SYN, FIN, CWR, and ECE) for various purposes
  - **Urgent data pointer**: indicates the location of the last byte of urgent data

→ Round-Trip Time Estimation and Timeout

$$\texttt{EstimatedRTT} = \texttt{EstimatedRTT} \cdot (1-\alpha) + \texttt{SampleRTT} \cdot \alpha$$
$$\texttt{DeviationRTT} = \texttt{DeviationRTT} \cdot (1-\beta) + |\,\texttt{SampleRTT} - \texttt{EstimatedRTT}\,| \cdot \beta$$
$$\texttt{Timeout} = \texttt{EstimatedRTT} + 4 \cdot \texttt{DeviationRTT}$$

Nota: Valores típicos para: $\alpha = 0.125 = 1/8$ [RFC 6298], $\beta = 0.25 = 1/4$.

→ Reliable Data Transfer

Com base no discutido na secção **Reliable Data Transfer**, o protocolo utilizado pelo TCP é o **temporizador/janela e ACKs cumulativos**, possuindo também uma forma rudimentar de controlo de congestão que passa pela duplicação dos intervalos de *timeout*: "(...) each time TCP retransmits, it sets the next timeout interval to twice the previous value [sofre um crescimento exponencial] (...) However, whenever the timer is started after data is received from the application above or an ACK is received, the `TimeoutInterval` is derived from the most recent values of `EstimatedRTT` and `DeviationRTT`."[1]

[§]"Note that `EstimatedRTT` is a weighted average of the `SampleRTT` values. (...) this weighted average puts more weight on recent samples than on old samples. This is natural, as the more recent samples better reflect the current congestion in the network. In statistics, such an average is called an **exponential weighted moving average (EWMA)**."[1]

→ TCP Flow Control

TCP flow control prevents sender-induced buffer overflow at the receiver, ensuring efficient data transmission. It uses a sliding window mechanism to adjust the transmission rate according to the receiver's buffer capacity and network conditions.

1. **Receiver buffer and advertised window (`rwnd`)**: The receiver allocates a buffer, with spare room defined as:

   $$\texttt{rwnd} = \texttt{RcvBuffer} - [\texttt{LastByteRcvd} - \texttt{LastByteRead}]$$

   The sender is informed of `rwnd` via the Window field in TCP headers.
2. **Sliding window and flow control operation**: The sender adjusts its sending window based on `rwnd` and network conditions. It stops transmitting data if `rwnd` drops to zero and resumes when a non-zero `rwnd` value is received.
3. **Handling zero `rwnd`**: If `rwnd` is zero and the receiver has no data or ACKs to send, the sender continues sending one-byte segments to keep the receiver updated about its buffer state.
4. **Interaction with congestion control**: Flow control works with congestion control mechanisms like slow start and congestion avoidance to adapt the transmission rate to both the receiver's buffer capacity and network bandwidth.

→ TCP Connection Management

The connection management process is mainly driven by a three-way handshake and a four-way handshake for connection establishment and termination, respectively.

1. **Three-way handshake (Connection Establishment)**: The handshake process involves three steps:
   (a) SYN: The client sends a TCP segment with the SYN (synchronize) flag set, indicating a connection request.
   (b) SYN + ACK: The server responds with a segment with both the SYN and ACK (acknowledge) flags set, acknowledging the client's request and synchronizing its own sequence number.
   (c) ACK: The client sends an ACK segment, confirming the establishment of the connection.
2. **Data transmission**: Once the connection is established, data transmission occurs using the sliding window and congestion control mechanisms.
3. **Four-way handshake (Connection Termination)**: The termination process involves the following steps:
   (a) FIN: The endpoint initiating the connection termination sends a segment with the FIN (finish) flag set, signaling the desire to close the connection.
   (b) ACK: The other endpoint acknowledges the FIN segment with an ACK segment.
   (c) FIN: The other endpoint also sends a FIN segment to indicate it is ready to close the connection.
   (d) ACK: The initiating endpoint acknowledges the second FIN segment with an ACK segment, completing the termination process.

[§]**TCP Connection States**: During connection management, both client and server transition through various states such as LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSING, TIME-WAIT, CLOSE-WAIT, LAST-ACK, and CLOSED.

↳ Congestion control

→ Approaches to Congestion Control

At the highest level, we can distinguish among congestion-control approaches by whether the network layer provides explicit assistance to the transport layer for congestion-control purposes:

- **Network-assisted congestion control:** In this approach, the network layer provides explicit feedback to the transport layer about the presence of congestion. The transport layer then adjusts its sending rate accordingly. Examples of this approach include ATM's ABR (Available Bit Rate) service and Frame Relay's BECN (Backward Explicit Congestion Notification) mechanism.
- **End-to-end congestion control:** In this approach, the network layer does not provide explicit feedback to the transport layer. Instead, the transport layer uses implicit feedback, typically from dropped packets or increased end-to-end delay, to infer the presence of congestion. The transport layer then adjusts its sending rate accordingly. The congestion-control mechanism used by TCP is an example of end-to-end congestion control.

Regardless of the approach taken, most congestion-control algorithms have the following objectives:

- **Maximize network utilization:** The algorithm should aim to make the best use of available network resources, ensuring that network capacity is fully utilized whenever possible.
- **Fairness:** The algorithm should allocate resources fairly among competing flows, preventing any one flow from monopolizing the network.
- **Stability:** The algorithm should be able to react to changes in network conditions and maintain stable performance, avoiding oscillations in throughput or other undesirable behavior.

→ Overview TCP

TCP sender limits its send rate using a congestion window (`cwnd`), which constrains the amount of unacknowledged data in the network. TCP detects congestion through loss events, either timeouts or three duplicate ACKs. In a congestion-free network, acknowledgments indicate successful data delivery, allowing the sender to increase `cwnd` and its send rate. TCP is self-clocking since it adjusts its congestion window size based on the rate of incoming acknowledgments:

$$\boxed{\texttt{LastByteSent} - \texttt{LastByteAcked} \leqslant \min\{\texttt{cwnd}, \texttt{rcvwnd}\}}$$

1. A lost segment implies congestion, and the sender should decrease its rate (by decreasing `cwnd`).
2. An acknowledged segment indicates the network is delivering data, so the sender can increase its rate.
3. **Bandwidth probing**: TCP increases its rate until a loss event occurs, then decreases the rate and probes again.

→ Classic TCP congestion control

(1) Slow Start:

- Initial `cwnd` is 1 MSS.
- Sender doubles sending rate every RTT.
- Exponential growth ends in three cases:
  - Timeout event: sets `cwnd` to 1 and `ssthresh` to `cwnd`/2 and starts slow start anew.
  - `cwnd` equals `ssthresh`: transitions into congestion avoidance mode.
  - Three duplicate ACKs: fast retransmit and enters fast recovery state.

(2) Congestion Avoidance:

- `cwnd` is approximately half its previous value.
- Increases `cwnd` by 1 MSS every RTT: common approach is to increase `cwnd` by MSS/`cwnd` when a new acknowledgment arrives.
- Linear increase (1 MSS per RTT) ends in two cases:
  - Timeout event: set `cwnd` to 1 MSS, update `ssthresh` to half the value of previous cwnd.
  - Triple duplicate ACK event: halve `cwnd` value, update `ssthresh` to half the value of previous `cwnd`, and enter fast-recovery state.

(3) Fast Recovery ("Fast recovery is a recommended, but not required, component of TCP"[1])

- In fast recovery, increase `cwnd` by 1 MSS for every duplicate ACK received for the missing segment.
- Upon receiving an ACK for the missing segment, enter the congestion-avoidance state and deflate `cwnd`.
- If a timeout event occurs, perform the same actions as in slow start and congestion avoidance:
  - Set `cwnd` to 1 MSS.
  - Set `ssthresh` to half the value of `cwnd` when the loss event occurred.
- After the timeout event, transition to the slow-start state.

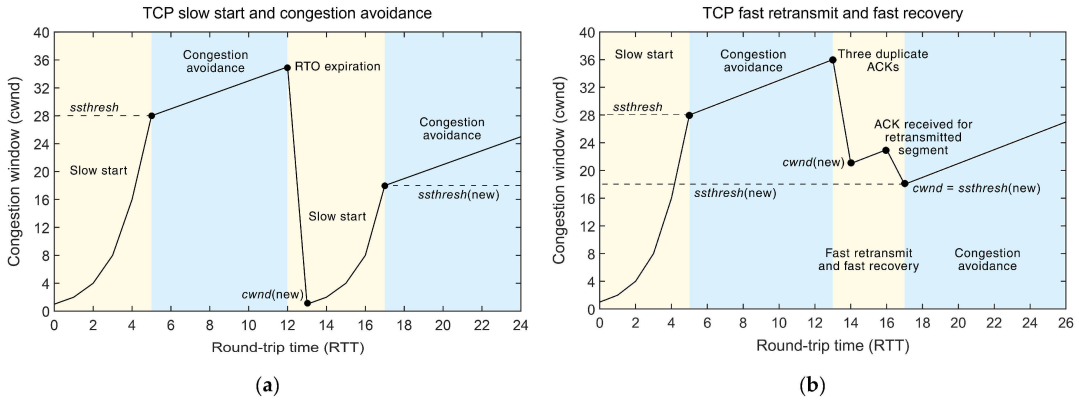


Fig. 28: Standard working principles of the TCP CC mechanism related to the interdependence between `cwnd` and RTT for: (a) the TCP slow start and congestion avoidance mechanism; (b) the TCP fast retransmit and fast recovery mechanism. [Lorincz 2021]

"For this reason, TCP congestion control is often referred to as an **additive-increase, multiplicative-decrease (AIMD)** form of congestion control."[1]

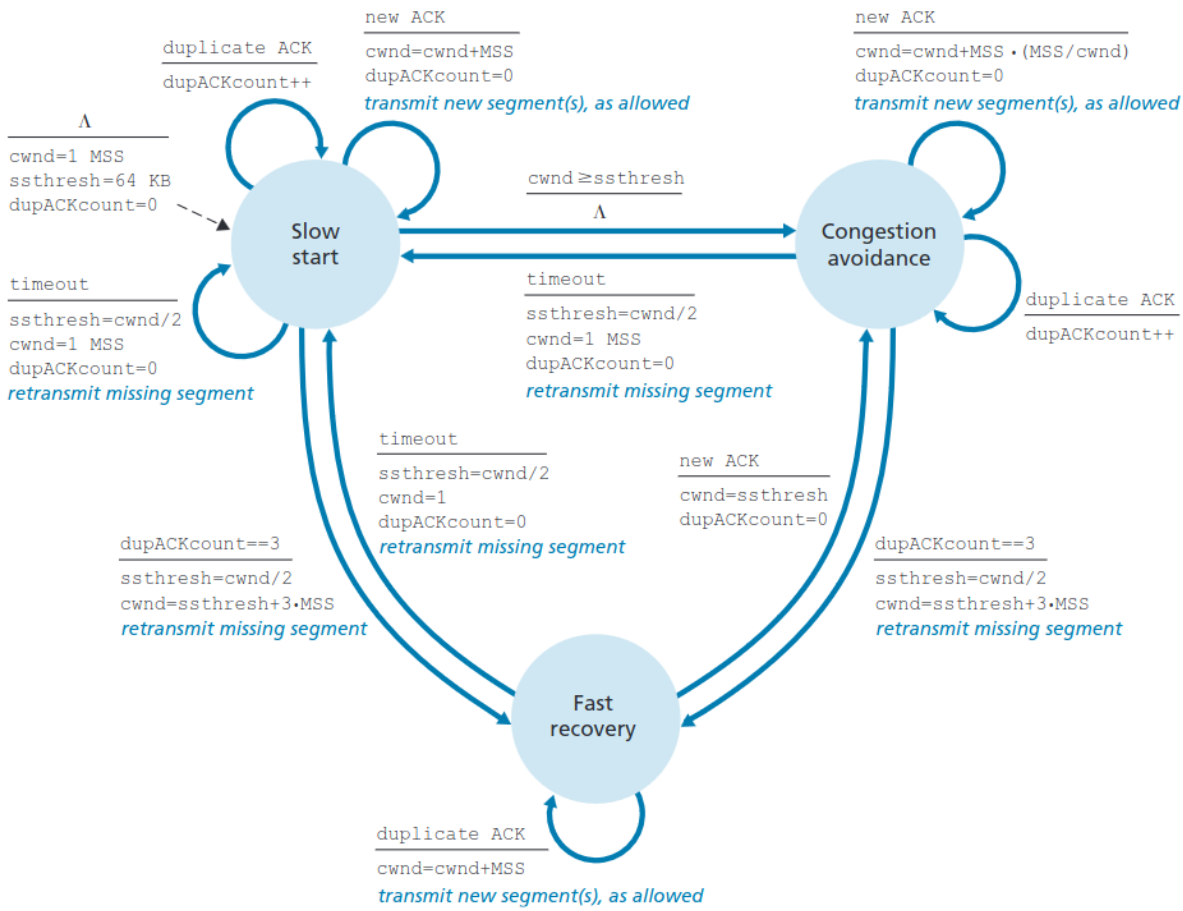


Fig. 29: Diagrama de controlo de congestão TCP [1]

⋆ Nota histórica: TCP Tahoe & TCP Reno

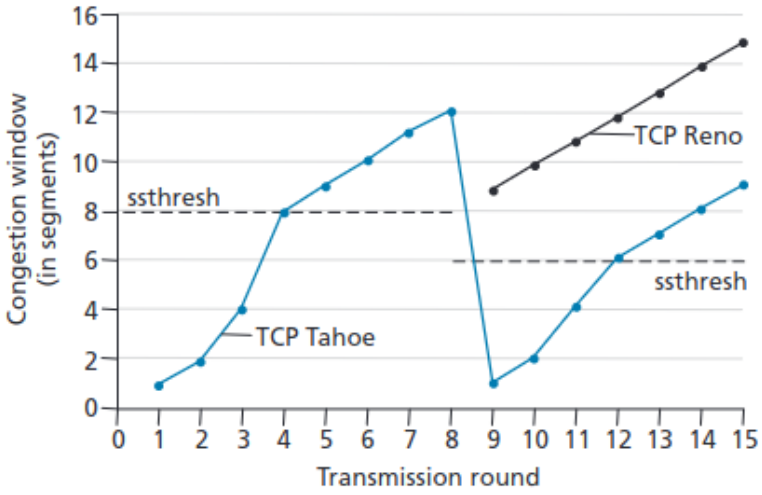


Fig. 30: "Evolution of TCP's congestion window (Tahoe and Reno)"[1]

"Fast recovery is a recommended, but not required, component of TCP [RFC 5681]. It is interesting that an early version of TCP, known as TCP Tahoe, unconditionally cut its congestion window to 1 MSS and entered the slow-start phase after either a timeout-indicated or triple-duplicate-ACK-indicated loss event. The newer version of TCP, TCP Reno, incorporated fast recovery."[1]

→ Extensions and Alternatives to TCP

- **RED (Random Early Discard):** An active queue management strategy that randomly drops packets before the buffer is full, encouraging sources to reduce sending rates and preventing global synchronization of flows.
- **ECN (Explicit Congestion Notification):** A mechanism allowing routers to signal congestion to end-points by setting a flag in the packet header, allowing end-points to adjust their sending rates without waiting for packet loss as a signal.
- **TCP Vegas:** A congestion control algorithm that detects congestion early by monitoring the difference between expected and actual throughput, adjusting sending rates to avoid oscillations and maintain steady performance.
- **TCP BBR (Bottleneck Bandwidth and Round-trip propagation time):** A congestion control algorithm using the bottleneck bandwidth and round-trip time to estimate available network capacity, achieving higher throughput and lower latency than traditional loss-based algorithms.
- **CUBIC TCP:** A congestion control algorithm using a cubic function for window growth, enabling faster recovery from congestion and better performance in high-bandwidth, high-latency networks.
- **QUIC (Quick UDP Internet Connections):** A transport layer protocol developed by Google using UDP, offering faster connection establishment, built-in encryption, and improved congestion control compared to TCP, resulting in better web performance over high-latency networks.

→ Congestion control fairness

Fairness in congestion control refers to the equal distribution of available bandwidth among concurrent connections. Various factors can affect the fairness of congestion control mechanisms:

- **TCP's AIMD algorithm:** For TCP's AIMD algorithm, it converges to provide an equal share of a bottleneck link's bandwidth among competing TCP connections, as demonstrated by Chiu 1989. This algorithm allows TCP connections to adjust their transmission rates based on network congestion and fairly share the available bandwidth.
- **Varying RTTs:** Multiple connections sharing a common bottleneck with varying RTTs can lead to unequal bandwidth allocation. Connections with smaller RTTs can grab available bandwidth more quickly, resulting in higher throughput than connections with larger RTTs.
- **UDP vs. TCP:** Many multimedia applications often run over UDP to avoid TCP's rate throttling, leading to a potentially unfair distribution of resources. UDP sources can maintain constant transmission rates and occasionally lose packets, while TCP sources reduce their rates during congestion. As a result, UDP sources can crowd out TCP traffic, causing fairness concerns.
- **Real-world scenarios:** In real-world situations, the ideal conditions such as only TCP connections traversing the bottleneck link, connections having the same RTT value, and only a single TCP connection being associated with a host-destination pair are rarely met. As a result, these factors can lead to unequal bandwidth allocation among different connections in client-server applications.

4. Camada de Rede

↳ Visão Geral

"The primary role of the network layer is deceptively simple—to move packets from a sending host to a receiving host."[1]

→ Network Layer Functions

Major functions of the network layer include:

1. **Addressing:** Assigning unique IP addresses to devices for identification and communication within a network.
2. **Routing:** Selecting the best path for data transmission between devices based on various routing algorithms.
3. **Forwarding:** Transferring data packets from a router's input to the appropriate output based on the destination address.
4. **Packetization:** Encapsulating data into packets for transmission, including fragmentation and reassembly when necessary.
5. **Error control:** Detecting and correcting transmission errors.

→ Forwarding and Routing: The Data and Control Planes

- **Data Plane:**
  The data plane is responsible for forwarding packets from one router to another. It refers to the router-local action of transferring a packet from an input link interface to the appropriate output link interface. This function is usually performed at very short timescales (typically a few nanoseconds) and is typically implemented in hardware.
- **Control Plane:**
  The control plane is responsible for determining the routes or paths taken by packets as they flow from a sender to a receiver. Routing algorithms are implemented in the control plane of the network layer. These algorithms operate at longer timescales (typically seconds) and are often implemented in software.
  1. **Traditional Approach:**
     In the traditional approach, both forwarding (data plane) and routing (control plane) functions are contained within a router. A routing algorithm runs in each router and communicates with routing algorithms in other routers to compute the values for its forwarding table.
  2. **SDN Approach:**
     In the Software-Defined Networking (SDN) approach, the control-plane routing functionality is separated from the physical router. The routing device performs forwarding only, while a remote controller computes and distributes forwarding tables. The controller is implemented in software and might be managed by the ISP or some third party.

§"A key element in every network router is its **forwarding table**. A router forwards a packet by examining the value of one or more fields in the arriving packet's header, and then using these header values to index into its forwarding table. The value stored in the forwarding table entry for those values indicates the outgoing link interface at that router to which that packet is to be forwarded."[1]

↳ Control Plane

→ Routing Protocols

⋆ Link-State Protocol (centralized routing algorithm)

The link-state protocol discovers adjacency, broadcasts network topology, and implements Dijkstra's algorithm for the shortest paths. It involves:

- **Adjacency discovery:** Routers exchange "hello" messages, discovering neighbors and establishing adjacencies (HELLO protocol).
- **Topology broadcast:** Routers create a **Link-State Advertisement (LSA)** containing adjacencies and broadcast it across the network.
- **Topology synchronization:** Routers maintain a **Link-State Database (LSDB)** storing LSAs, ensuring an identical and up-to-date network topology view.
- **Shortest path computation:** Routers use the LSDB and Dijkstra's algorithm to compute the shortest path tree, then populate forwarding tables for destination-based forwarding.

Let us define the following notation:

- `D(v)`: cost of the least-cost path from the source node to destination `v` as of this iteration of the algorithm.
- `p(v)`: previous node (neighbor of `v`) along the current least-cost path from the source to `v`.
- `N'`: subset of nodes; `v` is in `N'` if the least-cost path from the source to `v` is definitively known.

"Link-State (LS) Algorithm for Source Node `u`"[1]

```
initialization:
     N' = {u}
     for all nodes v
          if v is a neighbor of u
               then D(v) = c(u,v)
          else D(v) = +∞

loop {
     find w not in N' such that D(w) is a minimum
     add w to N'
     update D(v) for each neighbor v of w and not in N':
          D(v) = min(D(v), D(w)+ c(w,v))
     /* new cost to v is either old cost to v or known
      * least path cost to w plus cost from w to v */
} until N' = N
```

Pros & Cons:

Pros	Cons
Faster convergence, $\mathcal{O}(N^2)$, and more reliable path selection.	Higher memory and processing requirements.
Each router has a consistent view of the entire network topology.	More complex and harder to troubleshoot.
Resilient to routing loops.	May not scale well in very large networks.

Tab. 4: Pros & cons: Link-State Protocol.

Exemplo de execução do Link-State algorithm:

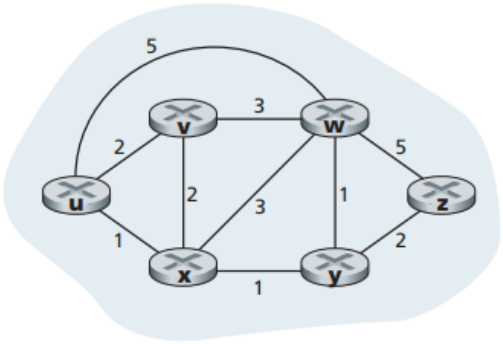
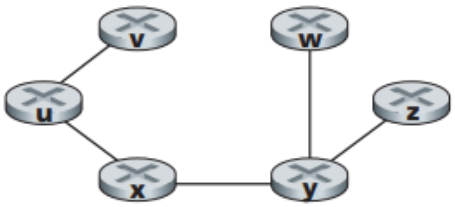


Fig. 31: "Abstract graph model of a computer network"[1]

step	*N'*	*D(v), p(v)*	*D(w), p(w)*	*D(x), p(x)*	*D(y), p(y)*	*D(z), p(z)*
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Fig. 32: "Running the link-state algorithm on the network"[1]



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Fig. 33: "Least cost path and forwarding table for node u"[1]

⋆ Distance-Vector Protocols (decentralized routing algorithm)

Bellman-Ford equation: $\boxed{D_x(y) = \min_v\{\, c(x,v) + D_v(y)\,\}}$

Distance-vector protocols employ the Bellman-Ford algorithm to compute the shortest path. They involve:

- **Routing table exchange:** Routers periodically exchange routing tables with neighbors, updating their tables accordingly.
- **Route selection:** Routers use the Bellman-Ford algorithm to update routing tables and determine the best path to each destination.
- **Count to infinity problem:** Slow convergence may lead to routing loops and suboptimal paths. "Split horizon" and "poison reverse" techniques can mitigate this problem.

"Distance-Vector (DV) Algorithm"[1]

```
initialization:
    for all destinations y in N:
        Dx(y) = c(x,y) /* if y is not a neighbor then c(x,y)= +∞ */
    for each neighbor w
        Dw(y) = ? for all destinations y in N
    for each neighbor w
        send distance vector Dx = [Dx(y): y in N] to w

loop {
    wait until
        I see a link cost change to some neighbor w
    or until
        I receive a distance vector from some neighbor w

    for each y in N:
        Dx(y) = min_v{c(x,v) + Dv(y)}

    if Dx(y) changed for any destination y
        send distance vector Dx = [Dx(y): y in N] to all neighbors
} forever
```

Addressing Routing Loops in Distance-Vector Protocols:

- **Routing loops** occur when a packet is forwarded in a continuous loop between routers, causing degraded network performance and packet loss. They often result from incorrect routing information due to slow convergence or the count-to-infinity issue in distance-vector protocols.
- The **count-to-infinity** issue arises when routers incrementally update their routing tables with outdated information, causing the routing metric to increase indefinitely. This can lead to suboptimal routing decisions and loops.
- **Split horizon** prevents routing loops by not propagating routing information learned from a neighbor back to that same neighbor, avoiding the spread of incorrect routing information.
- **Poison reverse** mitigates routing loops by advertising unreachable destinations with an infinite metric, informing neighbors not to use the advertising router as a path to the destination.
- Combining **split horizon** and **poison reverse** techniques can effectively prevent and resolve routing loops, leading to a more stable and efficient network.

Pros & Cons:

Pros	Cons
Lower computational requirements compared to link-state protocols. Simpler to configure and maintain.	Slower convergence, $\mathcal{O}(N \cdot E)$, leading to temporary routing loops and suboptimal paths. Limited network visibility. May not scale well in large networks due to the periodic exchange of routing tables.

Tab. 5: Pros & cons: Distance-Vector Protocols.

Exemplo de execução do Distance-Vector algorithm:

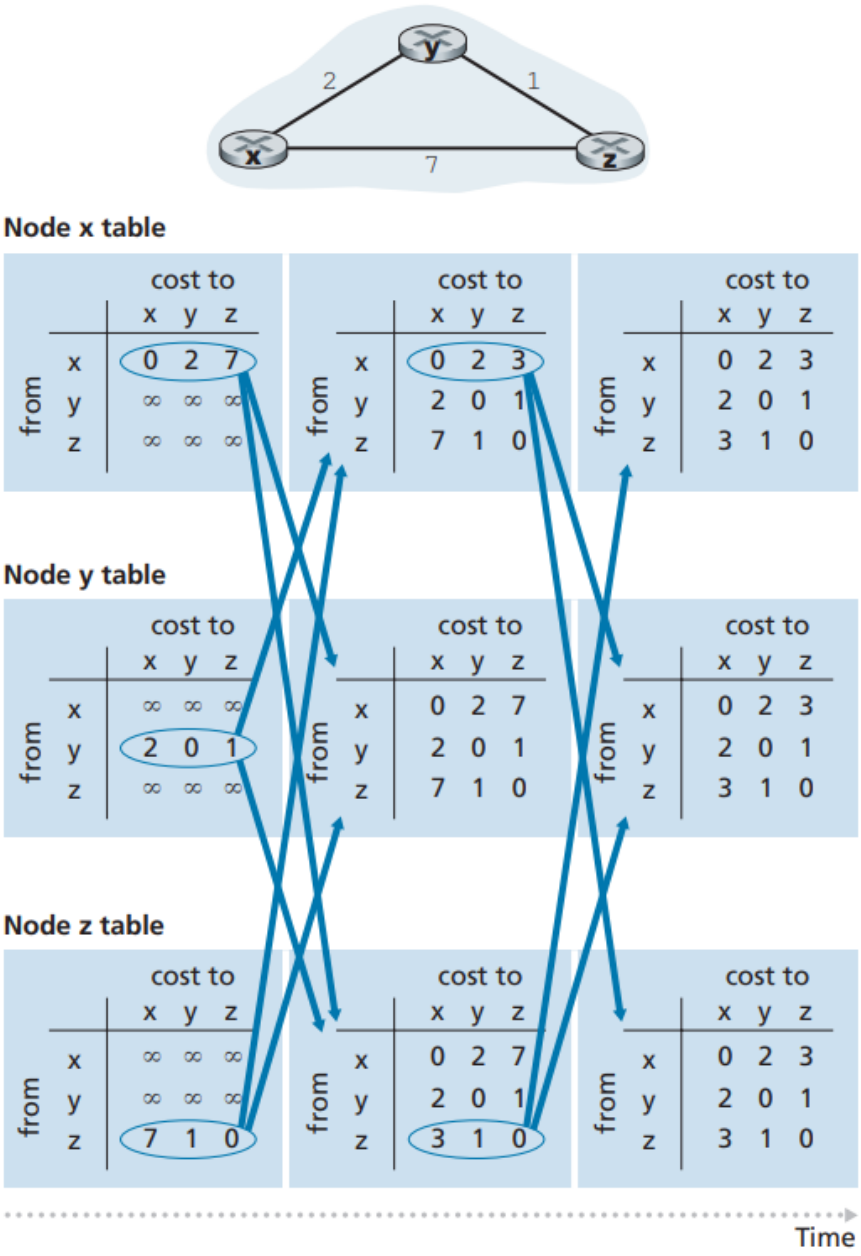


Fig. 34: "Distance-vector (DV) algorithm in operation"[1]

⋆ Path-Vector Protocols

Path-vector routing protocols extend distance-vector protocols by sharing the entire path information to a destination. They involve:

- **Path advertisement:** Routers advertise the entire path to each destination, instead of just the distance.
- **Path selection:** Routers use advertised paths to select the best path to each destination based on criteria such as path length, reliability, or administrative preferences.
- **Preventing routing loops:** Routers detect and prevent routing loops by checking for their own presence in the advertised path.

Pros & Cons:

Pros	Cons
More accurate and reliable path selection compared to distance-vector protocols.	Path advertisement can increase message size and bandwidth consumption.
Prevents routing loops by detecting and discarding paths containing the router's own ID.	Requires more memory and processing power than distance-vector protocols, but less than link-state protocols.
Supports policy-based routing by allowing administrators to define routing preferences.	Less straightforward than distance-vector protocols in terms of configuration and troubleshooting.

Tab. 6: Pros & cons: Path-Vector Protocols.

→ Hierarchical Routing and Autonomous Systems

Hierarchical routing simplifies large-scale network routing by organizing the topology into hierarchical levels, reducing routing information, and improving scalability and convergence times.

- **Autonomous Systems (ASes):** Collections of IP networks and routers controlled by a single organization with a common routing policy. The Internet is divided into thousands of ASes, each with a unique AS number.
- **Levels:** Hierarchical routing divides networks into tiers, such as core and edge, or additional levels like regional or local tiers.
- **Addressing:** Each hierarchy level has a unique address space, enabling efficient address aggregation and route summarization.
- **Route Aggregation and Summarization:** Higher-level routers maintain summarized routes, reducing the number of routes to store and exchange, and conserving resources.
- **Isolation:** Hierarchical routing isolates failures and updates within specific hierarchy levels or ASes, preventing network-wide instability.

Hierarchical routing is employed in both intradomain and interdomain routing, with OSPF and BGP using area or AS hierarchies. Intradomain routing within an AS uses protocols like OSPF, while interdomain routing between ASes is achieved through BGP.

→ Intra-AS Routing in the Internet

⋆ RIP: Routing Information Protocol

RIP is a distance-vector routing protocol that uses hop count as the metric for selecting the best path. It has a maximum hop count limit of 15, which restricts its scalability. Operates on top of UDP.

- **RIP Inner Workings**: RIP routers exchange their routing tables every 30 seconds. Upon receiving an update, a router will compare the received routes with its own routing table and update its entries if a shorter path is found.

⋆ OSPF: Open Shortest Path First

OSPF is a link-state routing protocol that calculates the shortest path using Dijkstra's algorithm. It is hierarchical and supports areas, allowing for better scalability compared to RIP.

- OSPF is a widely used, open, link-state routing protocol for intra-AS routing.
- Routers construct a complete topological map of the AS and use Dijkstra's algorithm to determine shortest-path trees.
- Link costs can be configured by the network administrator, OSPF does not dictate how they are set.
- Routers broadcast routing information to all routers in the AS, not just their neighbors.
- Link-state information is broadcasted when a link's state changes or at least once every 30 minutes for robustness.
- OSPF messages are carried by IP with an upper-layer protocol number of 89.
- The OSPF protocol implements functionality for reliable message transfer, link-state broadcast, link health check, and database synchronization with neighbors.

OSPF provides:

- **Security:** OSPF supports authentication (simple and MD5) to prevent unauthorized routers from injecting incorrect information.
- **Multiple same-cost paths:** OSPF allows load balancing over multiple equal-cost paths.
- **Integrated support for unicast and multicast routing:** Multicast OSPF (MOSPF) extends OSPF to support multicast routing.
- **Support for hierarchy within a single AS:**
  - OSPF AS can be divided into areas, each running its own link-state routing algorithm.
  - Area border routers handle routing between areas.
  - One OSPF area is designated as the backbone area, which routes traffic between other areas.

→ Inter-AS Routing in the Internet

⋆ BGP: Border Gateway Protocol

BGP is a path-vector routing protocol designed for inter-domain routing between Autonomous Systems (ASes). It is crucial for the global Internet routing infrastructure.

BGP provides each router a means to:

- Obtain prefix reachability information from neighboring ASes (allows each subnet to advertise its existence).
- Determine the "best" routes to the prefixes.

Advertising BGP Route Information: BGP routers, also known as "peers" or "neighbors," exchange routing updates containing the entire AS-path information. BGP uses various attributes, such as AS-path length and local preference, to select the best path.

- **eBGP:** External BGP is used to exchange routing information between ASes.
- **iBGP:** Internal BGP is used to exchange routing information within the same AS.

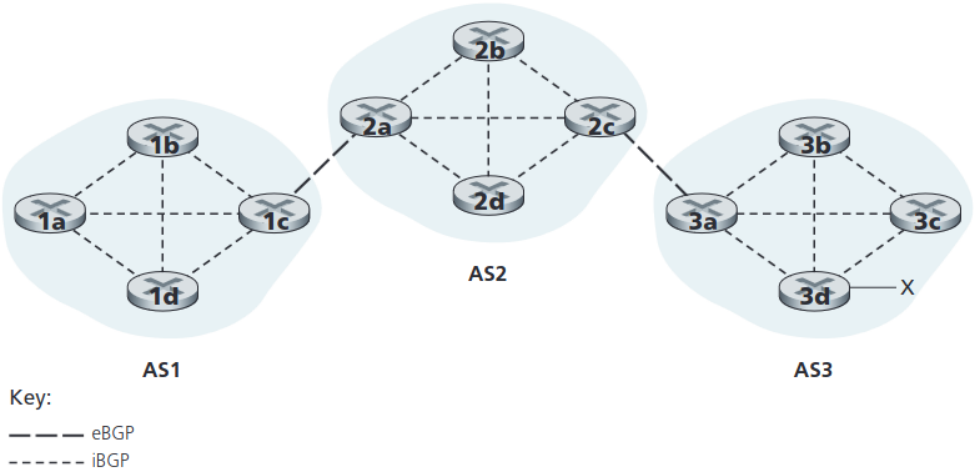


Fig. 35: eBGP and iBGP connections. Notice that iBGP connections do not necessarily follow the physical link between routers, in order to propagate the reachability information, both iBGP and eBGP sessions are used.

[§]**Commercial Routing Between Autonomous Systems**

In the context of commercial routing, ASes can establish various types of relationships, such as transit, peering, or customer-provider relationships. These relationships dictate how routing information is exchanged and propagated through the Internet.

- **Transit:** A transit AS provides connectivity for other ASes to reach the global Internet. Transit providers typically charge for their services.
- **Peering:** In a peering relationship, two ASes agree to exchange routing information without charging each other. This is commonly established between ISPs of similar size for mutual benefit.
- **Customer-Provider:** In this relationship, a provider AS offers Internet access and routing services to a customer AS. The customer typically pays the provider for the services.

Route Selection in BGP

Determining the Best Routes in BGP involves considering multiple paths from a given router to a destination subnet. BGP routers often receive reachability information about dozens of different possible paths. A router chooses among these paths and configures its forwarding table accordingly. When a router advertises a prefix across a BGP connection, it includes several BGP attributes, such as `AS-PATH` and `NEXT-HOP`, with the prefix.

- **AS-PATH** contains the list of ASes traversed by the advertisement. Routers use the `AS-PATH` attribute to detect and prevent looping advertisements by rejecting advertisements containing their own AS in the path list.
- **NEXT-HOP** is the IP address of the router interface that begins the `AS-PATH`. It serves as the critical link between inter-AS and intra-AS routing protocols.
- **LOCAL-PREF** is a BGP attribute representing the local preference value assigned to a route within an AS. This value is not passed outside the AS and is used to select the preferred exit point from the AS for a given route. The higher the local preference value, the more preferred the route.
- **MED** (Multi-Exit Discriminator) is another BGP attribute used to influence the choice of entry point to an AS for incoming traffic. The `MED` value suggests the preferred entry point by comparing the `MED` values of different routes advertised by neighboring ASes. The lower the `MED` value, the more preferred the route.

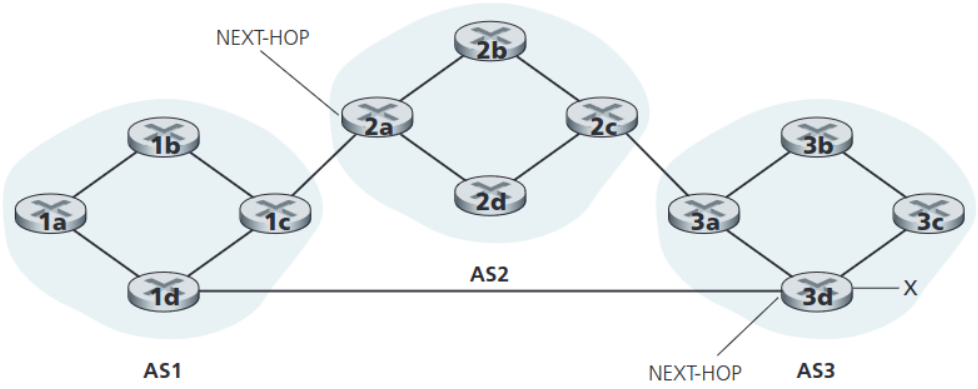


Fig. 36: The IP address of the leftmost interface of router 3d and of router 2a are known as Next-Hop.

Hot Potato Routing Algorithm

Hot potato routing, also known as "early-exit" or "closest-exit" routing, is a strategy used to select the best path based on the shortest internal distance within an AS. Hot potato is a selfish algorithm: it tries to reduce the cost in its own AS while ignoring the other components of the end-to-end costs outside its AS.

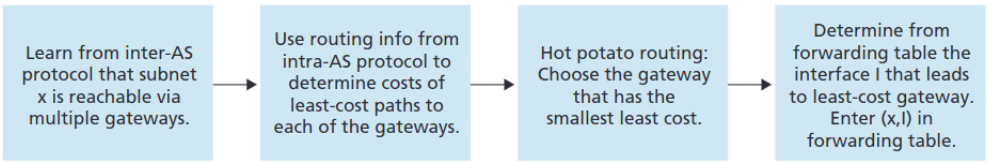


Fig. 37: Hot potato algorithm [1]

BGP Route-Selection Algorithm
In practice, BGP uses a more sophisticated route-selection algorithm that incorporates Hot Potato Routing and takes `LOCAL-PREF` and `MED` into account. For any given destination prefix, the input into BGP's route-selection algorithm is the set of all routes to that prefix that have been learned and accepted by the router. BGP sequentially invokes the following elimination rules until one route remains:

1. Select the routes with the highest local preference values (`LOCAL-PREF`).
2. From the remaining routes, select the route with the shortest `AS-PATH`.
3. From the remaining routes, choose the route with the lowest `MED` value.
4. From the remaining routes, use Hot Potato Routing to select the route with the closest `NEXT-HOP` router.
5. If more than one route still remains, use other BGP identifiers to select the route.

Route Filtering and Policy Control in BGP
BGP provides various methods for filtering and controlling route advertisements to implement routing policies.

- **Prefix Lists:** Used to filter routes based on the IP prefix.
- **AS-PATH Access Lists:** Used to filter routes based on the `AS-PATH` attribute.
- **Route Maps:** Used to apply complex policies that involve multiple BGP attributes. Route maps can manipulate attributes or filter routes based on various criteria.
- Each AS may choose to not advertise their routes to neighbouring ASes.
- Each AS attributes a preference value to the routes advertised by their neighbors.

By utilizing these filtering and policy control methods, BGP allows network administrators to manage inter-domain routing and implement policies based on commercial agreements or traffic engineering requirements.

- **AS routing policy can trump other considerations**, such as shortest AS path or hot potato routing; local-preference attribute value is fixed by the policy of the local AS.
- Access ISP[4] networks (last-mile/local ISPs) **advertise no paths to any other destinations except themselves**, ensuring all traffic leaving/entering them must have the source/destination within their own network.
- The rule of thumb followed by commercial ISPs: **any traffic flowing across an ISP's backbone network must have either a source or a destination (or both) in a network that is a customer of that ISP**.
- Differences between inter-AS and intra-AS routing protocols: Policy (dominates in inter-AS routing), Scale (critical for inter-AS routing, less so for intra-AS routing), and Performance (less important in inter-AS routing due to policy constraints).

[4]Access ISPs are usually connected to higher-tier ISPs, such as Tier-1 or Tier-2 ISPs, which have broader network coverage and provide backbone connectivity for lower-tier ISPs. In this sense, access ISPs are at the bottom of the ISP hierarchy, as they primarily focus on providing Internet services directly to the end-users.

↳ Data Plane

→ IPv4 Datagram Format

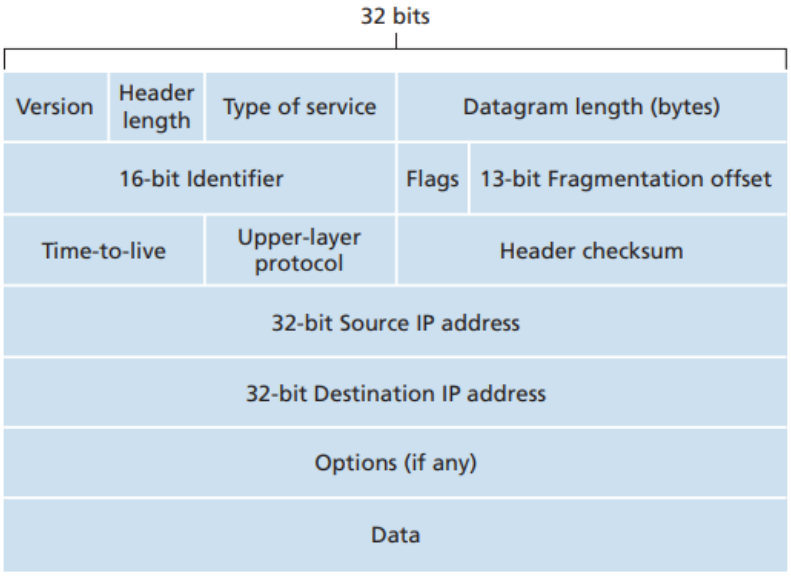


Fig. 38: IPv4 Datagram format [1]

- **Version number (4 bits):** Specifies the IP protocol version of the datagram, e.g., IPv4 or IPv6.
- **Header length (4 bits):** Indicates where the payload begins, considering variable number of options in the header.
- **Type of service (8 bits):** Distinguishes different types of IP datagrams, e.g., real-time vs non-real-time traffic, and supports Explicit Congestion Notification.
- **Datagram length (16 bits):** Total length of the IP datagram (header plus data) in bytes, with a theoretical maximum size of 65,535 bytes.
- **Identifier, flags, fragmentation offset (32 bits):** Related to IP fragmentation, where large datagrams are split into smaller ones, then reassembled at the destination.
- **Time-to-live (8 bits):** Ensures datagrams don't circulate indefinitely; decremented by one at each router, and dropped when reaching zero.
- **Protocol (8 bits):** Indicates the transport-layer protocol for the payload (e.g., 6 for TCP, 17 for UDP).
- **Header checksum (16 bits):** Helps detect bit errors in the received IP datagram using 1s complement arithmetic; recomputed and stored at each router.
- **Source IP address (32 bits):** IP address of the sender.
- **Destination IP address (32 bits):** IP address of the intended recipient.
- **Options (variable):** Allow IP header to be extended; used rarely and not included in IPv6 header.
- **Data (payload):** Contains the transport-layer segment (e.g., TCP or UDP) or other data types (e.g., ICMP messages) to be delivered to the destination.

→ IPv4 Addressing

Each IP address is 32 bits long (equivalently, 4 bytes). These addresses are typically written in so-called dotted-decimal notation, in which each byte of the address is written in its decimal form and is separated by a period (dot) from other bytes in the address:

`193.32.216.9` → `11000001 00100000 11011000 00001001`

⋆ Subnet and CIDR

Subnet

"To determine the subnets, detach each interface from its host or router, creating islands of isolated networks, with interfaces terminating the end points of the isolated networks. Each of these isolated networks is called a subnet."[1]

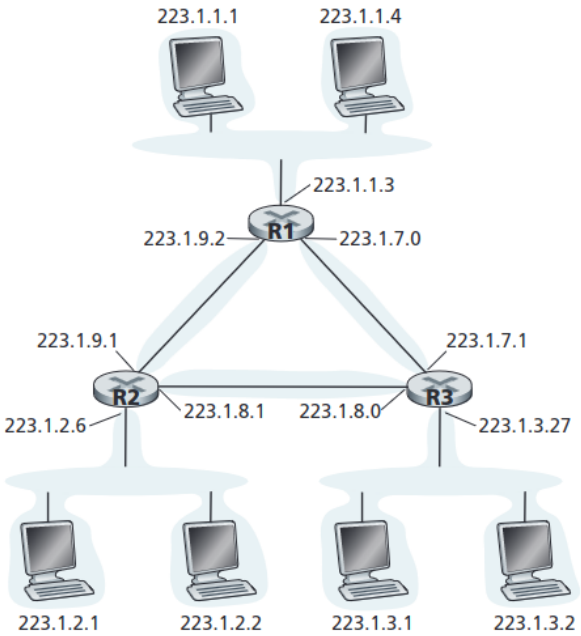


Fig. 39: 3 routers connected to 6 subnets

- The boundary between the host and the physical link is called an **interface**. A router may have more than one interface (In Fig. 39 each of the three routers possess 3 different interfaces)
- Multiple subnets can exist in a network, including point-to-point links connecting routers (see Fig. 39).
- An organization's network devices typically have IP addresses with a common network prefix.
- **Classless Interdomain Routing (CIDR)** generalizes subnet addressing by dividing the 32-bit IP address into two parts, with the network prefix indicated by x (**subnet mask**) in the a.b.c.d/x format (e.g.: `223.3.2.1/24` has prefix `223.3.2`).
- The x most significant bits are the network prefix, while the remaining 32-x bits can be used for subnetting within the organization.

Nota:
Address `255.255.255.255` is known has a brodcast address and is used to deliver a message to all hosts in the same subnet.

⋆ Allocating a block of addresses

- Network administrators obtain IP address blocks for their organizations from ISPs.
- ISPs can divide a larger allocated block into smaller blocks for their customers (e.g.: ISP's block `200.23.16.0/20` can be divided into eight smaller blocks for organizations such as `200.23.16.0/23`, `200.23.18.0/23` and so on).
- ISPs and other organizations can also obtain IP address blocks from a global authority, the **Internet Corporation for Assigned Names and Numbers (ICANN)**.
- ICANN manages IP addresses, DNS root servers, domain names, and domain name disputes, and allocates addresses to regional Internet registries.

→ The Dynamic Host Configuration Protocol (DHCP)

- Dynamic Host Configuration Protocol (DHCP) automates IP address assignment for hosts.
- Provides plug-and-play or zero-configuration functionality.
- Widely used in residential, enterprise, and wireless networks.
- DHCP operates as a client-server protocol.
- If no server is present on a subnet, a DHCP relay agent (typically a router) is needed.

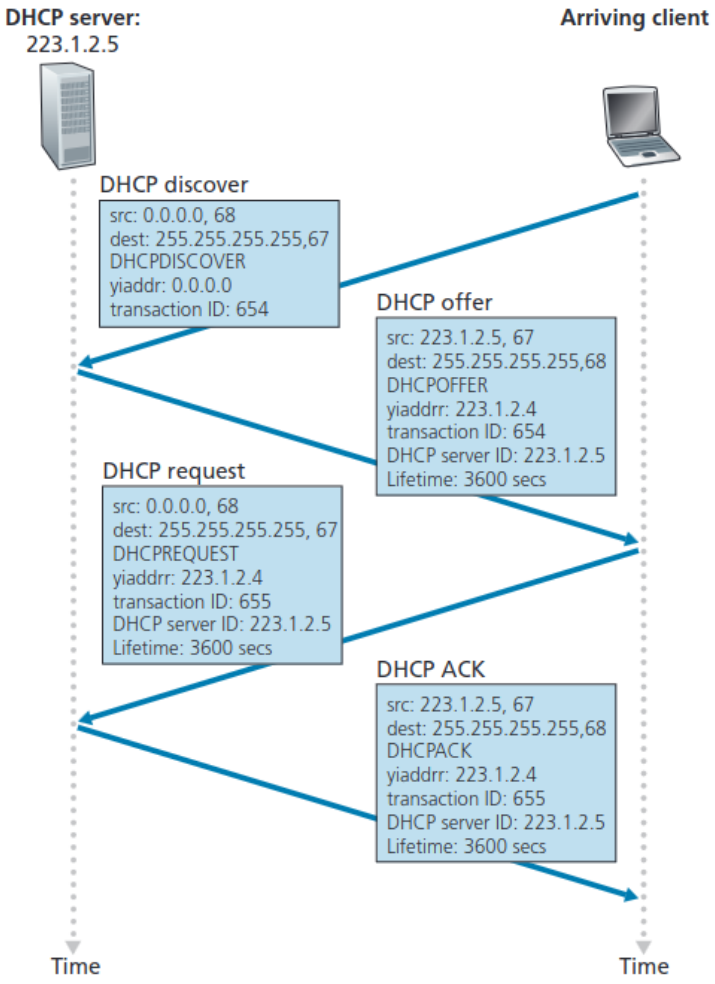


Fig. 40: Client-server DHCP protocol

1. **DHCP server discovery:** Client sends a DHCP discover message using the broadcast destination IP address `255.255.255.255` and source IP address `0.0.0.0`.
2. **DHCP server offer(s):** DHCP server responds with a DHCP offer message, also broadcast to all nodes on the subnet. Offer message contains transaction ID, proposed IP address, network mask, and IP address lease time.
3. **DHCP request:** Client chooses an offer and responds with a DHCP request message, echoing back the configuration parameters (usually DHCP request is broadcasted, however, the client can choose to unicast it directly to the DHCP server if the server's IP address is known, the same can be said for DHCP ACK).
4. **DHCP ACK:** Server responds with a DHCP ACK message, confirming the requested parameters.

→ Network Address Translation (NAT)

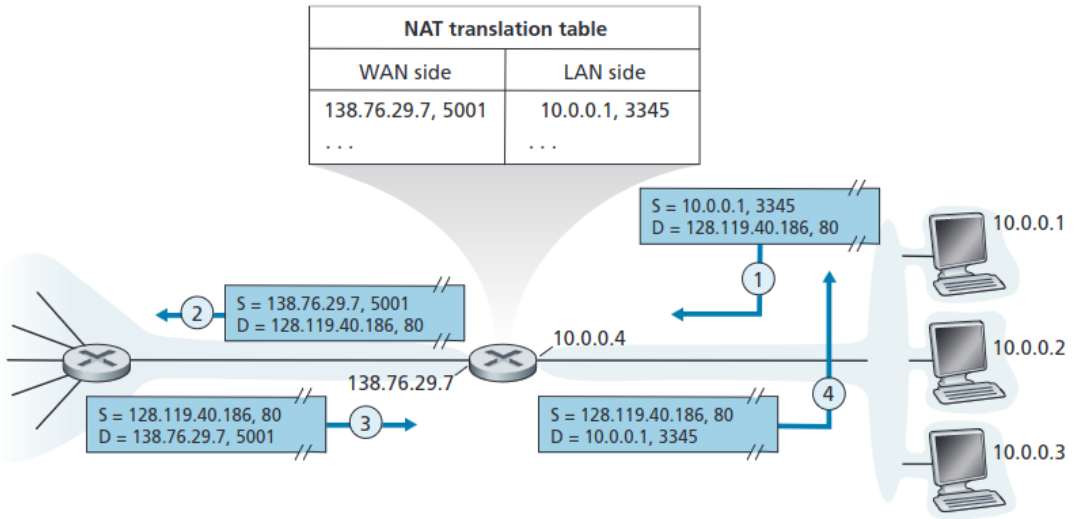


Fig. 41: Network address translation

The NAT-enabled router does not look like a router to the outside world. Instead the NAT router behaves to the outside world as a single device with a single IP address, possessing a **NAT translation table** capable of redirecting any messages to their specified destination through an (IP, PORT) association:

1. User requests a Web page from a Web server (port 80) with IP address `128.119.40.186`.
2. Host `10.0.0.1` assigns source port number 3345 and sends the datagram into the LAN.
3. NAT router generates a new source port number 5001, replaces the source IP address and port, and adds an entry to its NAT translation table.
4. Web server responds with a datagram **whose destination address is the NAT router's IP address**, and **destination port number is 5001**.
5. NAT router uses the NAT translation table to obtain the appropriate IP address and destination port number for the browser in the home network, rewrites the datagram's destination address and port, and forwards the datagram into the home network.

→ IPv6 Datagram Format

In the early 1990s, the Internet Engineering Task Force (IETF) began developing a successor to the IPv4 protocol due to concerns that the 32-bit IPv4 address space was running out. This led to the creation of the IPv6 protocol, which aimed to address the limitations of IPv4 and introduce improvements based on accumulated operational experience.

- **IPv6 Address Representation:**
  - IPv6 addresses are 128 bits long, separated into 8 fields of 16 bits each.
  - Each 16-bit field is represented by 4 hexadecimal characters.
- Example:
  - Original: `2001:0db8:130f:0000:0000:7000:0000:140b`
  - Simplified (without leading zeros): `2001:db8:130f:0:0:7000:0:140b`
  - Further simplified (with consecutive zero fields replaced by "::"): `2001:db8:130f::7000:0:140b`

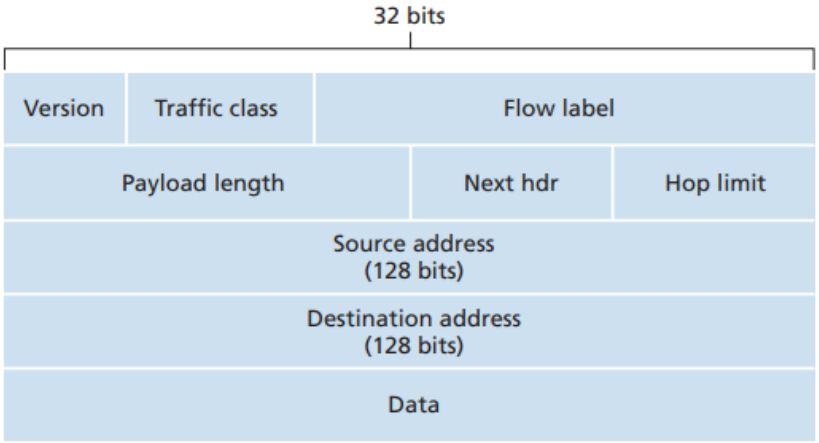


Fig. 42: IPv6 Datagram format [1]

- **Version (4 bits):** Identifies the IP version number; IPv6 carries a value of 6.
- **Traffic class (8 bits):** Prioritizes certain datagrams within a flow or from specific applications, similar to the Type of Service in IPv4.
- **Flow label (20 bits):** Used to identify a flow of datagrams, with an elusive definition; could distinguish traffic from different applications or users.
- **Payload length (16 bits):** Unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte datagram header.
- **Next header (8 bits):** Identifies the protocol to which the data field of the datagram will be delivered (e.g., TCP, UDP); same values as IPv4 protocol field.
- **Hop limit (8 bits):** Contents decremented by one at each router; if the count reaches zero, the router discards the datagram.
- **Source address (128 bits):** IPv6 address of the sender, described in RFC 4291.
- **Destination address (128 bits):** IPv6 address of the intended recipient, described in RFC 4291.
- **Data (payload):** Transport-layer segment (e.g., TCP, UDP) or other data types (e.g., ICMP messages) to be delivered to the destination.

IPv6 also omits certain fields that were present in the IPv4 datagram format:

- **Fragmentation/reassembly:** IPv6 does not allow fragmentation and reassembly at intermediate routers. If an IPv6 datagram is too large to be forwarded, the router drops the datagram and sends a "Packet Too Big" ICMP error message back to the sender, who can then resend the data using a smaller IP datagram size.
- **Header checksum:** Due to checksumming already being performed at the transport layer and link layer in the Internet, the header checksum was considered redundant and removed to speed up IP packet processing.
- **Options:** The options field is no longer a part of the standard IP header. Instead, it can be one of the possible next headers pointed to from within the IPv6 header, alongside TCP or UDP protocol headers. This change results in a fixed-length, 40-byte IP header.

⋆ Transitioning from IPv4 to IPv6

- New IPv6 systems can be made backward-compatible, but deployed IPv4 systems cannot handle IPv6 datagrams.
- A "flag day" for simultaneous transition from IPv4 to IPv6 is not feasible due to the vast number of devices.

Tunneling

- The most widely adopted approach for IPv4-to-IPv6 transition.
- IPv6 nodes wanting to communicate are connected by intervening IPv4 routers (forming a tunnel).
- The IPv6 node on the sending side encapsulates the entire IPv6 datagram into the payload field of an IPv4 datagram using the IPv6 protocol (number 41).
- The IPv4 routers in the tunnel route the IPv4 datagram without knowing it contains an IPv6 datagram, only aware of the IPv4 header information.
- The IPv6 node on the receiving side extracts the IPv6 datagram from the IPv4 datagram by recognizing protocol number 41 and routes it accordingly.

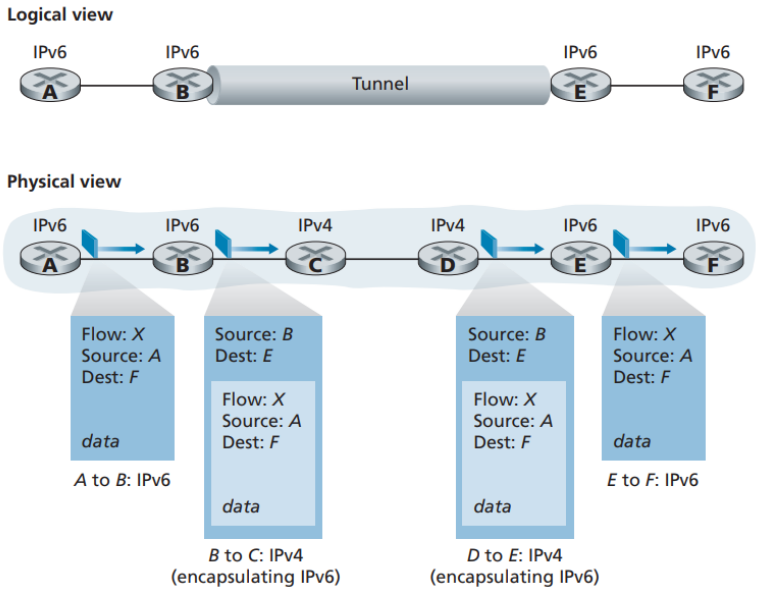


Fig. 43: Tunneling [1]

→ ICMP: The Internet Control Message Protocol

ICMP

ICMP is often considered part of IP, but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified as the upper-layer protocol (an upper-layer protocol number of 1), it demultiplexes the datagram's contents to ICMP, just as it would demultiplex a datagram's content to TCP or UDP.

ICMP messages consist of **type** and **code fields** and contains the **header of first 8 bytes of the triggering IP datagram**. ICMP messages are not only for error signaling; the ping program sends an ICMP echo request and the destination host responds with an ICMP echo reply. Traceroute is implemented with ICMP messages, sending a series of IP datagrams with increasing TTL values to identify routers and round-trip times between the source and destination hosts. ICMPv6 has reorganized the existing ICMP types and codes and added new ones for IPv6 functionality.

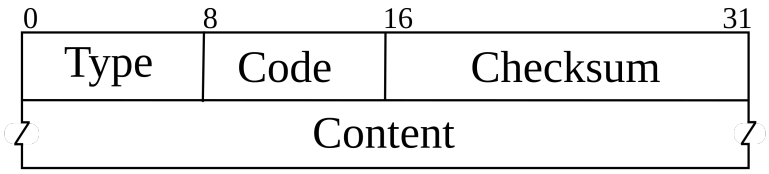


Fig. 44: ICMP header

ICMP Type	Code	Description
0	0	Echo reply (to ping)
3	0	Destination network unreachable
3	1	Destination host unreachable
3	2	Destination protocol unreachable
3	3	Destination port unreachable
3	6	Destination network unknown
3	7	Destination host unknown
4	0	Source quench (congestion control)
5	0	Redirect Datagram for the Network
5	1	Redirect Datagram for the Host
5	2	for the Type of Service and Network
5	3	for the Type of Service and Host
8	0	Echo request
9	0	Router advertisement
10	0	Router discovery
11	0	TTL expired
12	0	IP header bad

Tab. 7: ICMP message types [1]

ICMP redirects (types 5, codes 0-3) are used by routers to inform the sender that there is a more optimal route to the destination. When a router receives a packet and determines that the packet's next hop should be on the same interface it was received, the router sends an ICMP redirect message to the sender, indicating a more direct route to the destination. This can help optimize network traffic and reduce unnecessary load on routers.

Here's an example scenario:

1. Host A wants to send a packet to Host B.
2. Host A's routing table indicates that it should send the packet to Router 1.
3. Router 1 receives the packet, but based on its routing table, it determines that Host A should have sent the packet directly to Host B or to another router, Router 2, on the same network segment instead.
4. Router 1 sends an ICMP redirect message to Host A, notifying it that there is a more optimal route available to reach Host B.
5. Host A updates its routing table based on the received ICMP redirect message and sends subsequent packets to Host B using the new, more optimal route.

This process allows the network to self-optimize and avoid sending packets through unnecessary or suboptimal paths. However, it's worth mentioning that ICMP redirects are sometimes disabled for security reasons, as they can be exploited for man-in-the-middle attacks.

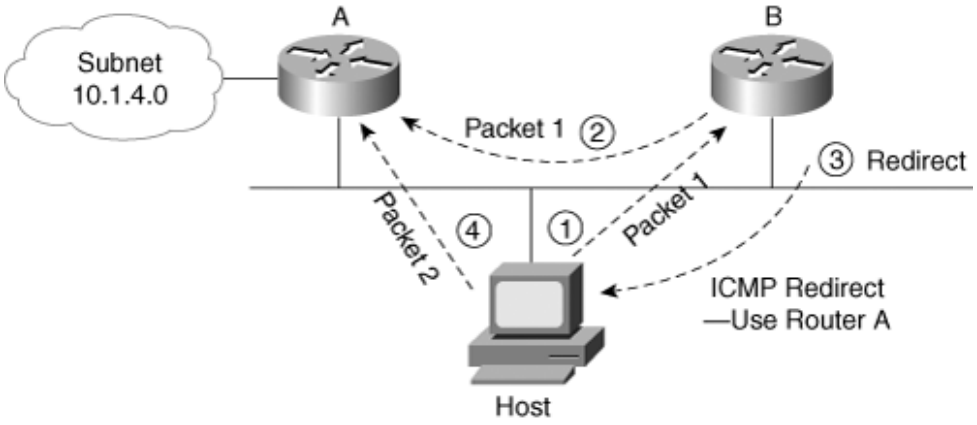


Fig. 45: ICMP redirect

5. Camada de Ligação de Dados

↳ Visão Geral

Link Layer

"The basic service of any link layer is to move a datagram from one node to an adjacent node over a single communication link."[1]

The link layer may provide the following services:

- **Framing:** Encapsulates network-layer datagrams in link-layer frames. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields.
- **Link access:** MAC (Medium Access Control) protocol governs frame transmission, coordinating multiple nodes on a broadcast link.
- **Reliable delivery:** Ensures error-free transmission across the link, useful for high-error-rate links but unnecessary for low-error-rate links.
- **Error detection and correction:** Identifies and corrects bit errors introduced by signal attenuation and electromagnetic noise.

The link layer implementation:

- Involves both hardware and software components.
- Uses a **network adapter**, also known as a **network interface controller (NIC)** for hardware implementation.
- Can be integrated into the motherboard chipset or implemented via a dedicated Ethernet chip.
- Handles various link layer services including framing, link access, and error detection.

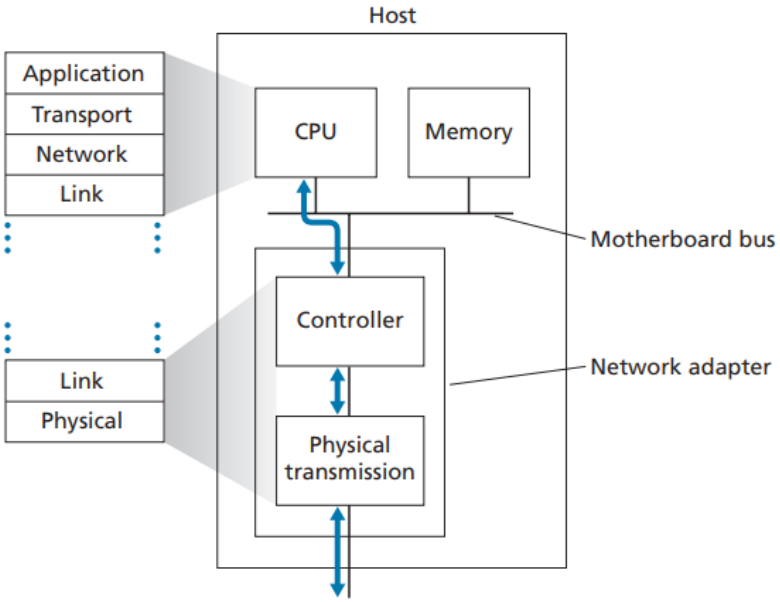


Fig. 46: "Network adapter: relationship to other host components and to protocol stack functionality"[1]

The link layer in the protocol stack involves both hardware and software components:

- On the sending side:
  - The network adapter takes a datagram created and stored in host memory by higher layers of the protocol stack.
  - It encapsulates the datagram in a link-layer frame.
  - The frame is transmitted into the communication link following the link-access protocol.
- On the receiving side:
  - The network adapter receives the entire frame.
  - It extracts the network-layer datagram.
  - Error detection is performed, if necessary.
- Higher-level link-layer features are implemented in software running on the host's CPU:
  - Software components assemble link-layer addressing information.
  - They activate the controller hardware.
  - Error conditions are handled, and datagrams are passed up to the network layer.

↳ Link Layer Error-Detection and -Correction Techniques

Error-detection and -correction techniques in the link layer are essential for maintaining data integrity during transmission between physically connected neighboring nodes. These techniques detect and correct bit errors in the link-layer frame.

→ Parity Checking

Parity checking is a basic error-detection technique that adds a single parity bit to the data being transmitted. The parity bit is set to make the total number of 1s in the data either even (even parity) or odd (odd parity). At the receiving end, the parity bit is checked to ensure that the total number of 1s is consistent with the selected parity scheme. If there is a discrepancy, an error is detected.

→ Checksumming

Checksumming is a more advanced error-detection technique that has been previously explained in the context of the transport layer for UDP and TCP. In this method, a checksum is calculated by the sender based on the data being transmitted and is then included with the transmitted data. At the receiving end, the receiver calculates the checksum based on the received data and compares it to the received checksum. If the calculated and received checksums do not match, an error is detected.

→ Cyclic Redundancy Checks (CRC)

Cyclic Redundancy Check (CRC) is an error-detection technique widely used in computer networks. It works by interpreting the bit string as a polynomial and performing modulo-2 arithmetic operations. The sender calculates CRC bits (`R`) and appends them to the data (`D`) to create a bit pattern divisible by a generator (`G`). The receiver checks the received data for errors by dividing the received bit pattern by `G` and verifying if the remainder is zero.

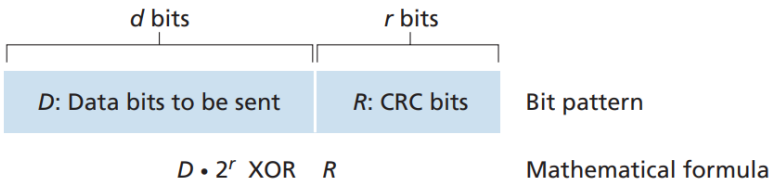


Fig. 47: CRC (Cyclic Redundancy Check) [1]

1. Choose a generator (`G`) with `r`+1 bits and make sure the leftmost bit of `G` is 1.
2. Calculate the CRC bits (`R`) by finding the remainder of the division $\mathtt{D} \cdot 2^{\mathtt{r}}/\mathtt{G}$, where `D` is the data bits.
3. Append the CRC bits (`R`) to the data (`D`) and send the d+r bit pattern.
4. The receiver divides the received $\mathtt{d} + \mathtt{r}$ bits by `G` using modulo-2 arithmetic.
5. If the remainder is nonzero, an error has occurred; otherwise, the data is accepted as correct.

> "International standards have been defined for 8-, 12-, 16-, and 32-bit generators (`G`). The CRC-32 32-bit standard, adopted in several link-level IEEE protocols, uses a generator:
>
> $$\mathtt{G}_{\text{CRC-32}} = \mathtt{100000100110000010001110110110111}$$
>
> Each CRC standard can detect burst errors of fewer than $\mathtt{r}+1$ bits, and under appropriate assumptions, a burst of length greater than $\mathtt{r}+1$ bits is detected with a probability of $1 - 0.5^{\mathtt{r}}$. Furthermore, each CRC standard can detect any odd number of bit errors."[1]

↳ Multiple Access Links and Protocols

Broadcast links allow multiple nodes to send and receive data on a shared channel. Multiple access protocols coordinate transmissions to minimize collisions and optimize throughput; they widely used in various network settings such as wired and wireless access networks, satellite networks, and local area networks (LANs). The three main categories of these protocols are channel partitioning, random access, and taking-turns protocols.

An ideal multiple access protocol should have the following characteristics:

1. When only one node has data to send, that node has a throughput of R bps.
2. When M nodes have data to send, each of these nodes has a throughput of R/M bps. This implies that each node should have an average transmission rate of R/M over a defined interval of time.
3. The protocol is decentralized, meaning there is no master node that represents a single point of failure for the network.
4. The protocol is simple, making it inexpensive to implement.

These protocols aim to efficiently manage transmissions and prevent bandwidth wastage in various network settings, including wired, wireless, and satellite networks.

→ Channel Partitioning Protocols

⋆ TDM (Time Division Multiplexing)

Brief: TDM divides time into time frames and further divides each time frame into N time slots. Each time slot is then assigned to one of the N nodes. Each node gets a dedicated transmission rate of R/N bps during each frame time.

Pros & Cons:

Pros	Cons
Eliminates collisions and is perfectly fair	Is limited to an average rate of R/N bps even when it is the only node with packets to send. A node must always wait for its turn in the transmission sequence

Tab. 8: Pros & cons: TDM

⋆ FDM (Frequency Division Multiplexing)

Brief: FDM divides the R bps channel into different frequencies (each with a bandwidth of R/N) and assigns each frequency to one of the N nodes. FDM thus creates N smaller channels of R/N bps out of the single, larger R bps channel.

Pros & Cons:

Pros	Cons
Avoids collisions and divides the bandwidth fairly among the N nodes.	Is limited to a bandwidth of R/N, even when it is the only node with packets to send.

Tab. 9: Pros & cons: FDM

⋆ CDMA (Code Division Multiple Access)

Brief: CDMA assigns a different code to each node. Each node then uses its unique code to encode the data bits it sends.

Pros & Cons:

Pros	Cons
If the codes are chosen carefully (as in, are ortogonal to each other), CDMA networks have the wonderful property that different nodes can transmit simultaneously and be successfully received by the receiver.	Transmissions on the same frequency with different codes are still received and decoded but simply re-appear as noise. This means the greater the number of users, the higher the noise level on the system

Tab. 10: Pros & cons: CDMA

→ Random Access Protocols

> "A transmitting node always transmits at the full rate of the channel, namely, R bps. When there is a collision, each node involved in the collision repeatedly retransmits its frame (that is, packet) until its frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. Instead it waits a random delay before retransmitting the frame."[1]

⋆ ALOHA

Consideremos o modelo:

- All frames consist of exactly L bits.
- Channel capacity of R bits/s
- Transmission of frames (new and retransmissions) at an average rate of g frames per second
- Average frame transmission rate per frame transmission time is $G = gL/R$ (offered load)
- All colliding frames at reception are lost
- The channel utilization S (throughput per frame time) is a function of G

O **processo de Poisson** é utilizado para modelar as interações deste protocolo.

$$\mathcal{P}r\{k \text{ transmissões durante o período de vulnerabilidade}\} = \frac{(gT)^k}{k!}\, e^{-gT}$$

em que T é o período de vulnerabilidade.

Pure ALOHA:

In pure ALOHA, a node transmits a frame immediately when it is ready. If a collision occurs, the node retransmits the frame with probability p or waits for another frame time with probability $1 - p$. The maximum efficiency of pure ALOHA can be derived by examining the probability of successful transmission for a single node. Let the frame transmission time be the unit of time. At any given time, the probability that a node is transmitting a frame is p. Suppose this frame begins transmission at time t_0. For the frame to be successfully transmitted, no other nodes can start transmitting in the interval of time $[t_0 - 1, t_0]$ and during the interval $[t_0, t_0 + 1]$. The probability that all other nodes do not begin a transmission in each interval is $(1-p)^{N-1}$, where N is the total number of nodes. Thus, the probability of a successful transmission for a given node is $p(1-p)^{2(N-1)}$. Taking the limit, we find that the maximum efficiency of pure ALOHA is $1/(2e)$, half of that of slotted ALOHA. This is the price for a fully decentralized ALOHA protocol.

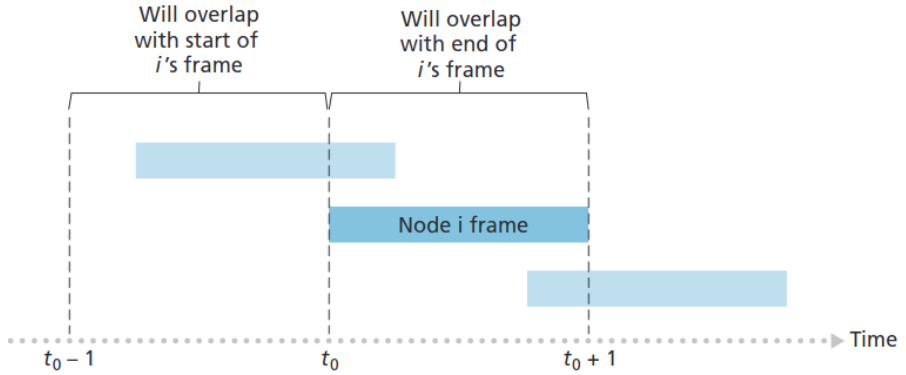


Fig. 48: Pure ALOHA vulnerability window. The **vulnerable period** is $T = 2L/R$—two frame times!

Slotted ALOHA:

With slotted ALOHA we'll assume the following:

- Time is divided into slots of size L/R (a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.

The operations of slotted ALOHA are the following:

- When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot.
- If there isn't a collision, the node has successfully transmitted its frame and thus need not consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.)
- If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability p until the frame is transmitted without a collision (as in, the node effectively tosses a biased coin; the event heads corresponds to "retransmit", which occurs with probability p.).

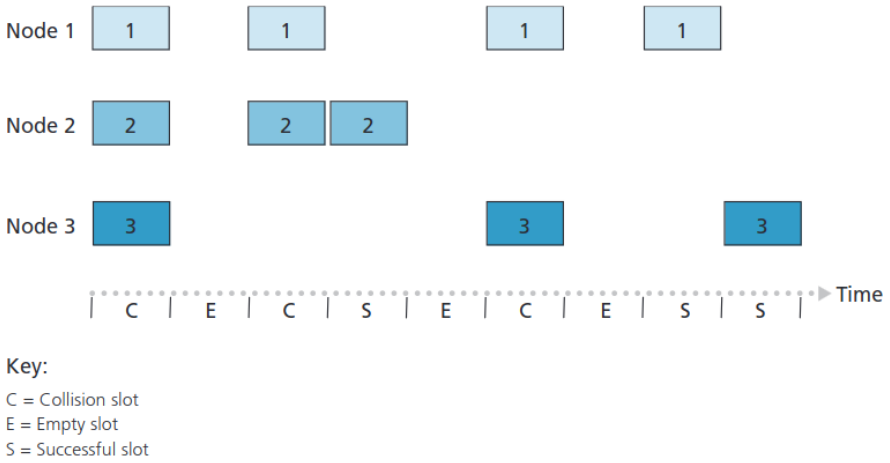


Fig. 49: Collision visualization in ALOHA protocol

Efficiency Comparison:

Definindo a utilização do canal como:

$$S(G) = Ge^{-\alpha G} \text{ em que } \alpha \propto \text{período de vulnerabilidade}$$

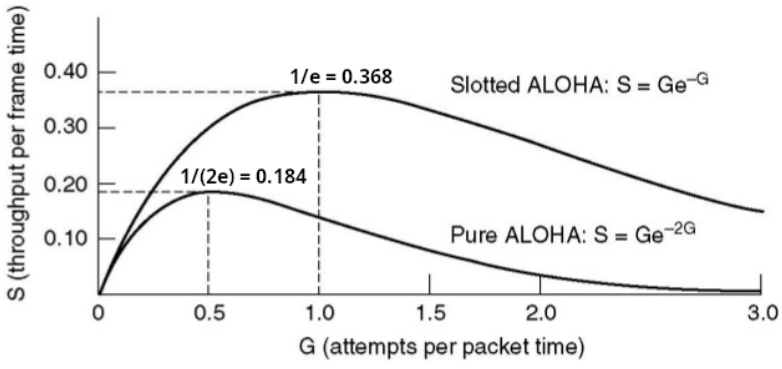


Fig. 50: ALOHA protocols—efficiency comparison.

⋆ CSMA (Carrier Sense Multiple Access)

In CSMA, a node's decision to transmit takes into account the activity of other nodes attached to the broadcast channel, unlike ALOHA protocols. Two important rules for CSMA are:

1. **Listen before speaking**: This is called carrier sensing, where a node listens to the channel before transmitting. If a frame from another node is being transmitted, the node waits until no transmissions are detected and then begins transmission.
2. **Stop talking if someone else begins talking**: In networking, this is called collision detection. A transmitting node listens to the channel while it is transmitting. If it detects an interfering frame, it stops transmitting and waits a random amount of time before repeating the sense-and-transmit-when-idle cycle.

Collisions can still occur in CSMA despite carrier sensing. Consider four nodes (A, B, C, D) attached to a linear broadcast bus. At time t_0, node B senses the channel is idle and begins transmitting. The bits propagate in both directions along the broadcast medium. At time t_1 ($t_1 > t_0$), node D has a frame to send. Even though B is transmitting, the bits haven't reached D yet, so D senses the channel idle at t_1. In accordance with the CSMA protocol, D begins transmitting, and shortly after, B's transmission interferes with D's transmission at D. It is evident that the end-to-end channel propagation delay plays a crucial role in determining the performance of a broadcast channel.

⋆ CSMA/CD (Carrier Sense Multiple Access with Collision Detection)

CSMA/CD improves upon CSMA by incorporating collision detection. Nodes abort their transmission shortly after detecting a collision, thus reducing wasted channel time. The operation of CSMA/CD from the perspective of a node's adapter is as follows:

1. Obtain datagram, prepare link-layer frame, put frame in adapter buffer.
2. Sense channel idle or busy; if idle, transmit frame; if busy, wait until no signal energy detected and transmit.
3. Monitor for signal energy from other adapters while transmitting.
4. If no signal energy detected during transmission, frame is complete; if signal energy detected, abort transmission.
5. Enter the exponential backoff state after aborting and return to step 2.

Binary Exponential Backoff: $\boxed{t_{\text{backoff}} = K \cdot t_{\text{frame}}}$

This algorithm addresses the problem of choosing an appropriate random backoff time interval by adapting the range of values based on the number of collisions experienced by a frame. When a frame has undergone n collisions, the algorithm selects the value of K randomly from $\{0, \ldots, 2^n - 1\}$. The maximum value for n is usually limited to 10 to prevent excessive waiting times.

CSMA/CD Efficiency:

Efficiency is defined as the long-run fraction of time during which frames are being transmitted without collisions when there is a large number of active nodes with many frames to send. Let d_{prop} be the maximum signal energy propagation time between two adapters, and d_{trans} be the time to transmit a maximum-size frame. The efficiency approximation is:

$$\text{Efficiency} = \frac{1}{1 + 5\, d_{prop}/d_{trans}}$$

⋆ CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)

CSMA/CA is an improvement to the CSMA protocol that focuses on collision avoidance. It is commonly used in wireless networks like Wi-Fi, where collision detection is not feasible due to hardware limitations and issues like the hidden terminal problem and fading. In wireless LANs, such as 802.11, CSMA/CA employs a random access protocol with a distributed coordination function (DCF) and incorporates inter-frame spacing intervals, such as **Distributed Inter-frame Space (DIFS)** and **Short Inter-frame Space (SIFS)**.

1. If the station senses the channel idle initially, it transmits its frame after a short period of time known as the Distributed Inter-frame Space (DIFS).
2. Otherwise, the station chooses a random backoff value using binary exponential backoff (as discussed earlier) and counts down this value after DIFS when the channel is sensed idle. The counter value remains frozen while the channel is sensed busy.
3. When the counter reaches zero (which can only occur while the channel is sensed idle), the station transmits the entire frame and then waits for an acknowledgment.
4. If an acknowledgment is received, the transmitting station knows that its frame has been correctly received at the destination station.
   (a) If the station has another frame to send, it begins the protocol at step 2.
   (b) If the acknowledgment isn't received, the transmitting station reenters the backoff phase in step 2, with the random value chosen from a larger interval.

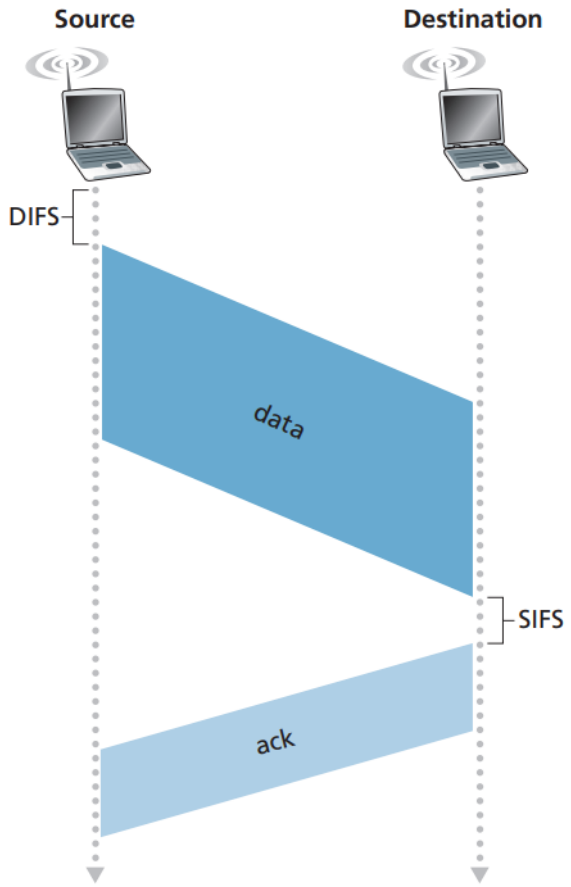


Fig. 51: Link-layer acknowledgments [1]

⋆ RTS-CTS (Request to Send - Clear to Send)

RTS-CTS is a mechanism used in conjunction with CSMA/CA to further reduce collisions in wireless networks, especially in the presence of hidden terminals.

Hidden terminal

The hidden terminal problem occurs when multiple wireless stations within the range of an access point (AP) are unable to detect each other's signals due to limited signal ranges. This situation results in collisions and wasted channel resources, as stations might transmit frames unknowingly during another station's ongoing transmission to the AP.

To tackle this problem, the IEEE 802.11 protocol introduces the use of short RTS and CTS control frames to reserve channel access. The sender transmits a **Request to Send (RTS)** packet to the receiver, which replies with a **Clear to Send (CTS)** packet. Neighboring nodes overhearing the RTS or CTS packets update their Network Allocation Vector, helping them anticipate when the channel will be available again.

The steps for RTS-CTS are:

1. The sender senses the channel to check if it is idle before initiating the transmission process.
2. The sender prepares to transmit a frame and sends an RTS packet to the receiver, specifying the total time required to transmit the frame and its acknowledgment.
3. The receiver, upon receiving the RTS packet, responds with a CTS packet, which confirms its readiness and reserves the channel for the specified duration.
4. Neighboring nodes that overhear the RTS or CTS packets update their Network Allocation Vector (NAV) to determine when the channel will be free again and defer their transmissions accordingly.
5. The sender waits for the CTS packet; if received, it proceeds to transmit the frame.
6. If the sender does not receive the CTS packet, it performs a binary exponential backoff and retries the process from step 1.
7. Once the frame is successfully transmitted, the receiver sends an ACK packet to the sender to confirm successful reception.
8. Other nodes that overhear the ACK packet can also update their NAV and defer their transmissions accordingly.

By employing the RTS-CTS mechanism, nodes reserve the channel, effectively notifying other nodes of ongoing transmissions. This helps to reduce collisions and address the hidden terminal problem. However, the RTS-CTS exchange can introduce additional delay and consume channel resources.

To strike a balance between collision avoidance and performance, the RTS-CTS mechanism is typically only used for reserving the channel for long `DATA` frames. In practice, wireless stations can set an RTS threshold, activating the RTS-CTS sequence only for frames exceeding the threshold. Many wireless stations adopt a default RTS threshold value larger than the maximum frame length, effectively skipping the RTS-CTS sequence for all `DATA` frames sent, optimizing resource usage and minimizing delay.

→ Taking-Turns Protocols

Taking-turns protocols, such as polling and token-passing protocols, aim to improve throughput and reduce collisions.

⋆ Polling Protocol

A master node polls other nodes in a round-robin[5] fashion, allowing them to transmit up to a maximum number of frames. This method eliminates collisions and empty slots, resulting in higher efficiency. However, it introduces polling delay and risks a single point of failure with the master node.

Example: Bluetooth Protocol

1. A master node (e.g., a Bluetooth headset) establishes connections with one or more slave nodes (e.g., a smartphone and a tablet).
2. The master node polls each slave node, granting them permission to transmit data for a limited time.
3. After a slave node has transmitted its data, the master node polls the next slave node in the sequence.
4. This process continues in a cyclic manner, enabling communication between the master and slave nodes.

⋆ Token-Passing Protocol

In token-passing protocols, a special-purpose frame (the token) is exchanged among nodes in a fixed order. A node holding the token can transmit frames before passing the token to the next node. This method is decentralized and efficient, but may face issues related to node failure or token circulation.

Example: IEEE 802.5 Token Ring Protocol

1. Nodes are organized in a logical ring topology, with each node connected to its two neighboring nodes.
2. The token, a special-purpose frame, circulates through the ring.
3. When a node receives the token and has data to transmit, it replaces the token with its data frame and sends it along the ring.
4. As the data frame traverses the ring, the destination node reads the data and marks the frame as acknowledged.
5. The frame eventually returns to the source node, which then removes the frame from the ring and releases a new token for circulation.
6. If a node receives the token but has no data to transmit, it forwards the token to the next node in the ring.

[5] In the context of polling protocols, round-robin refers to the method of visiting each node in a predefined cyclic sequence, ensuring that each node gets an opportunity to transmit data in a fair manner.

↳ MAC addressing and ARP protocol

A link-layer address is variously called a LAN address, a physical address, or a MAC address.

- The MAC address is 6 bytes long, giving 2^{48} possible MAC addresses.
- The Mac address is expressed in hexadecimal notation.
- MAC addresses were designed to be permanent (however there exists software which allows one to alter them).
- IEEE manages the MAC address space.
- Each adapter has it's own unique MAC address.
- An adapter's MAC address has a flat structure (as opposed to a hierarchical structure) and doesn't change no matter where the adapter goes.
- When an adapter receives a frame, it will check to see whether the destination MAC address in the frame matches its own MAC address. If there is a match, the adapter extracts the enclosed datagram and passes the datagram up the protocol stack, if not, the adapter discards the frame.

Nota: An adapter may choose to broadcast it's frame to all available adapters in the LAN. To do so it uses a **MAC brodcast address**: `FF-FF-FF-FF-FF-FF`.

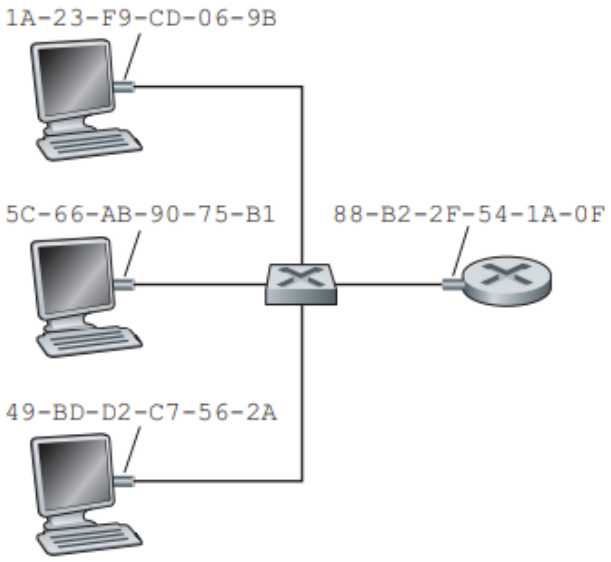


Fig. 52: Each interface connected to a LAN has a unique MAC address. Notice that the switch does not possess a MAC address in each of their interfaces, this is because the job of the link-layer switch is to carry datagrams between hosts and routers; a switch does this job transparently, that is, without the host or router having to explicitly address the frame to the intervening switch

MAC address spoofing is a technique where an attacker manipulates the unique hardware identifier (Media Access Control address) of a network interface, allowing them to impersonate other devices, bypass access controls, or evade monitoring on a network.

→ Address Resolution Protocol (ARP)

An ARP module in the sending host takes any IP address on the same LAN as input, and returns the corresponding MAC address of the destination. Each host and router has an **ARP table** in its memory, which contains mappings of IP addresses to MAC addresses. It also contains a time-to-live (TTL) value, which indicates when each mapping will be deleted from the table.

IP Address	MAC Address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Fig. 53: Example of an ARP table [1]

Note: ARP resolves IP addresses for hosts and router interfaces **on the same subnet**.

The ARP protocol is defined by the following steps:

1. The sending host checks if the ARP table contains an entry for the destination node's IP address.
2. If not, the sender constructs an ARP query packet containing the sending and receiving IP and MAC addresses.
3. The sender passes the ARP query packet to the adapter, indicating to send the packet to the MAC broadcast address (`FF-FF-FF-FF-FF-FF`).
4. The adapter encapsulates the ARP packet in a link-layer frame and transmits it into the subnet using the broadcast address.
5. All adapters on the subnet receive the frame and pass the ARP packet to their ARP modules.
6. The ARP modules check if their IP address matches the destination IP address in the ARP packet.
7. The module with a matching IP address sends a response ARP packet back to the querying host with the desired IP-to-MAC mapping.
8. The querying host updates its ARP table and sends its IP datagram, encapsulated in a link-layer frame with the destination MAC obtained from the ARP response.

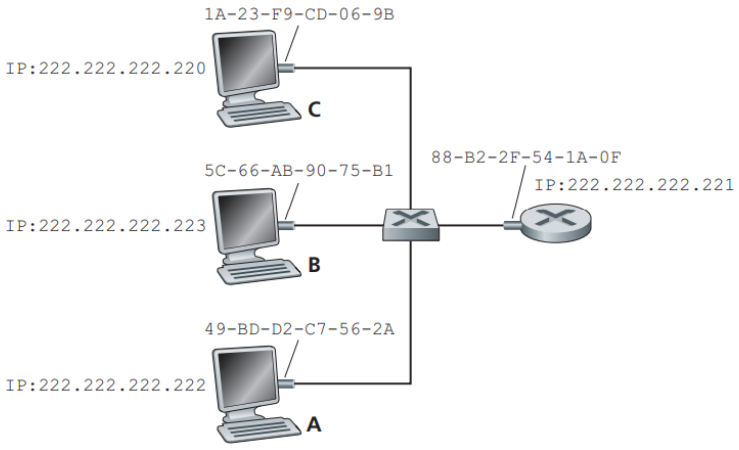


Fig. 54: Each interface on a LAN has an IP address and a MAC address [1]

⋆ Sending a Datagram off the Subnet

Sending a datagram off the subnet is more complicated than sending it within the subnet. Let's consider a network with two subnets interconnected by a router. In this scenario, each host has one IP address and one adapter, while a router has an IP address for each of its interfaces, as well as an ARP module and an adapter for each interface.

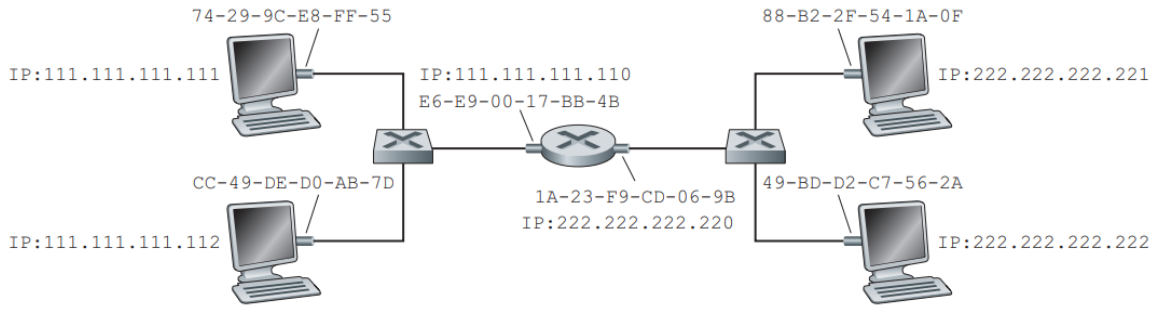


Fig. 55: Two subnets interconnected by a router [1]

Assume that host A wants to send an IP datagram to host B on a different subnet. The sender must pass the datagram to its adapter and indicate an appropriate destination MAC address. But, using the MAC address of the adapter for host B directly would be incorrect, as the datagram would not pass through any adapters on the sender's subnet[6].

Instead, the appropriate MAC address for the frame should be that of the router interface on the sender's subnet, which can be obtained using ARP. After creating a frame containing the datagram addressed to host B, the sender sends it to the router interface on its subnet. The router then forwards the datagram to the router interface on the destination subnet, which sends the frame into that subnet. Finally, the destination MAC address of the frame is indeed the MAC address of the ultimate destination (host B), which is also obtained via ARP.

The general behavior of sending datagrams off of a subnet to another can be summarized as follows:

1. The sender checks the destination IP address and uses ARP to obtain the MAC address of the router interface on its subnet, if needed.
2. The sender creates a frame containing the datagram addressed to the destination host, using the MAC address obtained in the previous step.
3. The sender's adapter transmits the frame onto its subnet; the router's adapter on the sender's subnet receives it and passes the datagram to the router's network layer.
4. The router consults its forwarding table, determines the correct output interface, and uses ARP to obtain the destination MAC address for the destination host or next-hop router interface.
5. The router's adapter on the destination subnet encapsulates the datagram in a new frame with the obtained MAC address and transmits it onto the destination subnet.
6. The destination host's adapter receives the frame and passes the datagram to its network layer.

[6]"The datagram would just die and go to datagram heaven."[1]

→ Ethernet

Ethernet is a widely used local area network (LAN) technology that provides a communication protocol for transmitting data packets over wired connections.

Ethernet has dominated the wired LAN market since the 1970s due to its early deployment, simplicity, cost-effectiveness, and continuous evolution in data rates. Initially using a coaxial bus topology, Ethernet transitioned to a hub-based star topology in the late 1990s, which still operated as a broadcast LAN. In the early 2000s, the hub was replaced with a switch, making Ethernet collision-less and transforming it into a store-and-forward packet switch that operates up to layer 2. This evolution has contributed to Ethernet's ongoing success and prevalence in local area networking.

Fig. 56: Ethernet frame structure [1]

- **Preamble** (8 bytes): serves to "wake up" the receiving adapters and synchronize their clocks to that of the sender's clock. The first 7 bytes have a value of 10101010, and the last byte is 10101011. The last two bits of the 8th byte alert the receiving adapter that the "important stuff" is about to come.
- **Destination address** (6 bytes): contains the MAC address of the destination adapter.
- **Source address** (6 bytes): contains the MAC address of the sending adapter.
- **Type field** (2 bytes): allows Ethernet to multiplex network-layer protocols, such as IP, Novell IPX, AppleTalk, and ARP, among others. Each protocol has its own standardized type number.
- **Data field** (46 to 1,500 bytes): carries the IP datagram, requires padding or fragmentation as needed. The maximum transmission unit (MTU) of Ethernet is 1,500 bytes. If the IP datagram exceeds 1,500 bytes, the host has to fragment the datagram. If the IP datagram is less than 46 bytes, the data field has to be "stuffed" to fill it out to 46 bytes.
- **Cyclic redundancy check (CRC)** (4 bytes): allows the receiving adapter to detect bit errors in the frame. The receiving adapter runs a CRC check but does not send an acknowledgment for passed frames nor a negative acknowledgment for failed frames.

Ethernet technologies provide an unreliable and connectionless service at the link layer, analogous to IP's layer-3 datagram service and UDP's layer-4 connectionless service. Ethernet has evolved into different flavors, such as

- `10BASE-T`: 10 Mbps over twisted-pair copper cables
- `10BASE-2`: 10 Mbps over thin coaxial cables
- `100BASE-T`: 100 Mbps over twisted-pair copper cables
- `1000BASE-LX`: 1 Gbps over long-wavelength single-mode fiber-optic cables
- `10GBASE-T`: 10 Gbps over twisted-pair copper cables
- `40GBASE-T`: 40 Gbps over twisted-pair copper cables

standardized by the IEEE 802.3 CSMA/CD (Ethernet) working group.

↳ Link-Layer Switches

> "The role of the switch is to receive incoming link-layer frames and forward them onto outgoing links"[1]

The switch is **transparent** to the hosts and routers in the subnet, that is, a host/router addresses a frame to another host/router (rather than addressing the frame to the switch) and happily sends the frame into the LAN, unaware that a switch will be receiving the frame and forwarding it. Switches may also encounter congestion (the rate at which frames arrive to any one of the switch's output interfaces may temporarily exceed the link capacity of that interface), for which **they also possess buffers** capable of storing queued frames.

Both routers and switches are **store-and-forward devices** with different functionalities:

- **Routers (encaminhadores):**
  - Operate at the network layer.
  - Determine optimal paths to destinations.
  - Require configuration.
  - Protect against broadcast storms.
- **Switches (comutadores):**
  - Operate at the data link layer.
  - Determine a spanning tree for all destinations.
  - Do not require configuration (plug-and-play).
  - Do not protect against broadcast storms.

→ Filtering and Forwarding

- **Filtering:** the switch function that determines if a frame should be forwarded or dropped.
- **Forwarding:** the switch function that identifies the interfaces to which a frame should be directed, and moves the frame to those interfaces (switches forward packets based on MAC addresses rather than on IP addresses).

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Fig. 57: Portion of a switch table

Filtering and forwarding actions may take one of the following scenarios:

1. **No table entry for destination address**: Switch broadcasts the frame to all interfaces except the input interface.
2. **Table entry associates destination with input interface**: Switch filters and discards the frame, as it doesn't need forwarding.
3. **Table entry associates destination with different output interface**: Switch forwards the frame to the specified output interface.

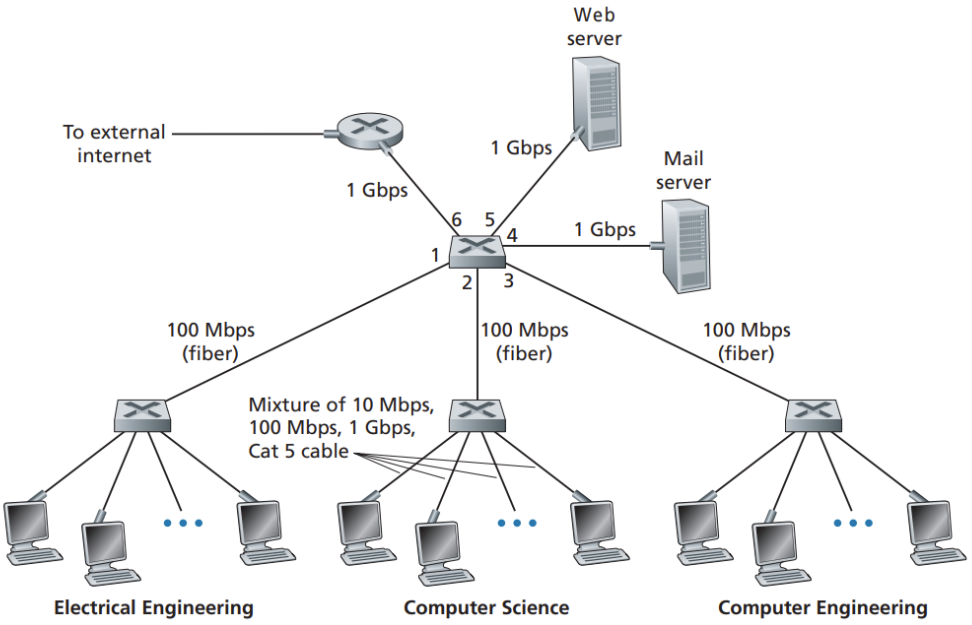


Fig. 58: Diagram of an example subnet

Taking into account the picture above:

- Frame with destination address `62-FE-F7-11-89-A3` arrives at switch from interface 1: Switch filters (discards) the frame, as it has already been broadcast on the LAN segment containing the destination.
- Frame with destination address `62-FE-F7-11-89-A3` arrives from interface 2: Switch forwards the frame to the output buffer preceding interface 1, as the destination is in that direction.
- A complete and accurate switch table ensures frame forwarding without unnecessary broadcasting.

→ Self-Learning

1. The switch table is initially empty.
2. For each incoming frame received on an interface, the switch stores the MAC address in the frame's source address field, the interface from which the frame arrived, and the current time. This way, the switch records the LAN segment on which the sender resides.
3. The switch deletes an address in the table if no frames are received with that address as the source address after some period of time (the aging time). This ensures that outdated MAC addresses are eventually purged from the switch table.

Taking Fig. 56 into consideration again:

> "Suppose at time 9:39 a frame with source address `01-12-23-34-45-56` arrives from interface 2. Suppose that this address is not in the switch table. Then the switch adds a new entry to the table. Continuing with this same example, suppose that the aging time for this switch is 60 minutes, and no frames with source address `62-FE-F7-11-89-A3` arrive to the switch between 9:32 and 10:32. Then at time 10:32, the switch removes this address from its table."[1]

→ Spanning Tree Protocol (STP)

The Spanning Tree Protocol (STP) is a link-layer protocol that ensures a loop-free topology in Ethernet networks by creating a spanning tree. It is particularly useful in bridged and switched networks, where loops can cause broadcast storms and result in duplicate frames. STP was standardized as IEEE 802.1D.

The primary objectives of STP are:

- Preventing loops in the network.
- Selecting a designated port for forwarding frames.
- Blocking other redundant ports to maintain a loop-free topology.

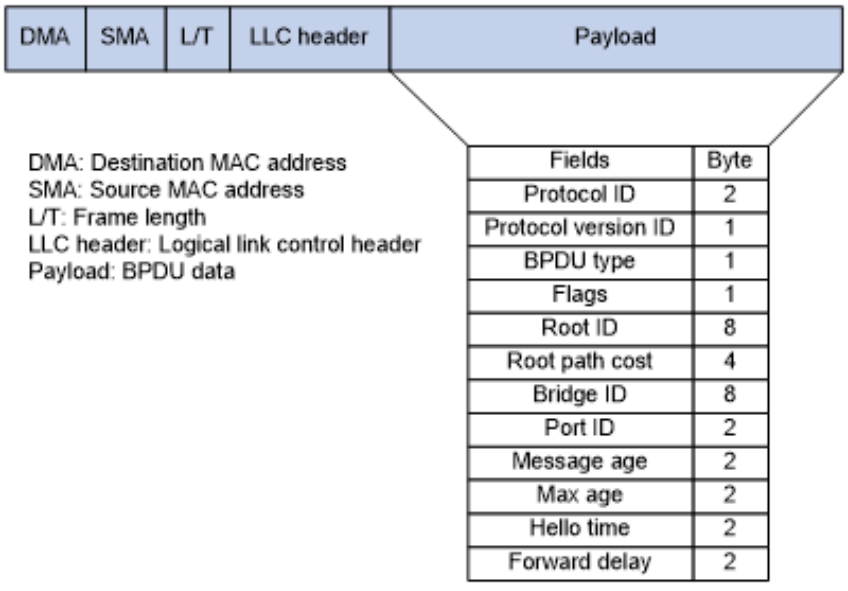
STP operates in the following steps:

1. Elect a root bridge by comparing Bridge IDs (BIDs) which consist of a bridge's priority value and MAC address.
2. Calculate the shortest path from each switch to the root bridge using path costs.
3. Select the port with the lowest path cost as the root port for each non-root bridge.
4. Elect designated ports on each network segment, selecting the port with the lowest path cost to the root bridge.
5. Block all other ports, putting them in a backup state, preventing loops.

During the operation of STP, switches exchange Bridge Protocol Data Units (BPDUs) to share information about network topology and maintain the correct spanning tree state.

Bridge Protocol Data Unit (BPDU)

BPDUs are special data packets that contain crucial information such as the switch's unique identifier, the sender's root bridge identifier, the sender's root path cost, and the sender's switch port identifier. BPDUs are used by STP to determine the best loop-free topology.



Fields	Byte
Protocol ID	2
Protocol version ID	1
BPDU type	1
Flags	1
Root ID	8
Root path cost	4
Bridge ID	8
Port ID	2
Message age	2
Max age	2
Hello time	2
Forward delay	2

Fig. 59: STP header [**?**]

→ Virtual Local Area Network (VLAN)

A **Virtual Local Area Network (VLAN)** is a network configuration that allows multiple virtual LANs to be defined over a single physical LAN infrastructure. This enables hosts within a VLAN to communicate as if they were exclusively connected to the switch, providing solutions for issues such as lack of traffic isolation, inefficient use of switches, and user management difficulties in modern institutional LANs.

In port-based VLANs, the switch's ports are divided into groups, with each group forming a VLAN and its own broadcast domain. VLAN switches can be interconnected using two methods: direct connections between VLAN ports or a more scalable approach called **VLAN trunking**. In trunking, a special port on each switch is configured as a trunk port that belongs to all VLANs. Frames sent to any VLAN are forwarded over the trunk link to the other switch.

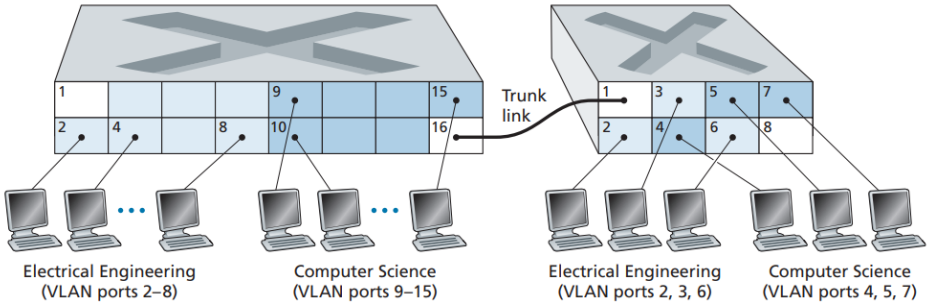


Fig. 60: Connecting two VLAN switches with two VLANs [1]

To identify the frame's VLAN membership, the IEEE 802.1Q frame format is used. It extends the standard Ethernet frame by adding a four-byte **VLAN tag** to the header, containing the **VLAN identifier** and a priority field. This tag is added and removed by the sending and receiving switches, respectively.

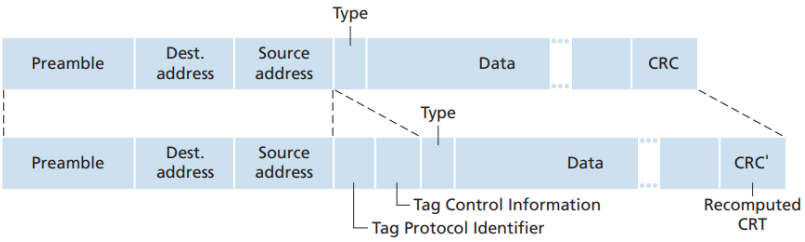


Fig. 61: Original Ethernet frame (top), 802.1Q-tagged Ethernet VLAN frame (below) [1]

Besides port-based VLANs, there are MAC-based VLANs where the network manager specifies the set of MAC addresses belonging to each VLAN. VLANs can also be defined based on network-layer protocols and other criteria. It is even possible for VLANs to be extended across IP routers, allowing geographically separated LANs to form a single VLAN.

↳ WiFi: 802.11 Wireless LANs

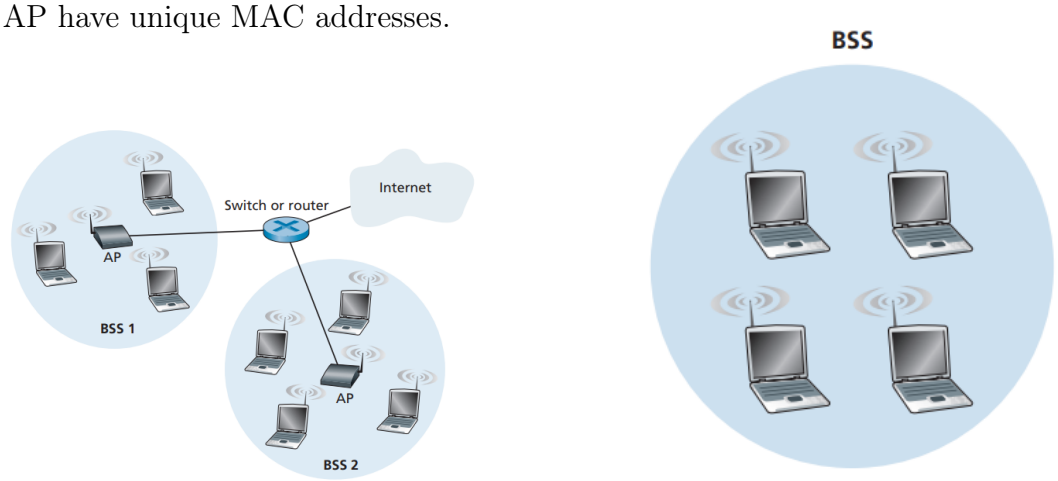
Wireless LANs (WLANs), particularly the IEEE 802.11 standard, have become ubiquitous in the modern world, providing internet access in various settings like homes, workplaces, and public areas. Despite having some physical layer differences, these standards share the 802.11 frame format, backward compatibility, and use the same medium access protocol, CSMA/CA. The 802.11ax standard also supports centralized scheduling.

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11 n (WiFi 4)	2009	600 Mbps	70 m	2.4, 5 Ghz
802.11 ac (WiFi 5)	2013	3.47 Gpbs	70 m	5 Ghz
802.11 ax (WiFi 6)	2020 (expected)	14 Gbps	70 m	2.4, 5 Ghz

Tab. 11: Summary of some IEEE 802.11 standards [1]

→ Wireless LAN Architecture

The basic building block of the 802.11 architecture is the basic service set (BSS), comprising one or more wireless stations and an access point (AP). APs connect to interconnection devices, which in turn connect to the Internet. Each wireless station and AP have unique MAC addresses.



(a) Infrastructure wireless LAN (rede estruturada) [1]

(b) An ad hoc network [1]

Fig. 62

- **Infrastructure wireless LANs** (Fig. 62 (a)):
  - Consist of wireless stations connected to an AP.
  - AP connects to the wired network infrastructure.
  - Centralized control, increased range, and access to wired resources.
  - Stations associate with APs by passively or actively scanning channels.
  - Device obtains IP address through DHCP.
- **Ad hoc networks** (Fig. 62 (b)):
  - Decentralized networks without an AP.
  - Wireless stations communicate directly with each other.
  - Useful for temporary, spontaneous, or small-scale communication.
  - Lacks centralized control and extended range of infrastructure networks.

[§]Ad hoc networks do not inherently have direct access to the Internet. However, if one device in the network has an Internet connection (e.g., via cellular network or separate Wi-Fi), it can potentially share its connection with other devices through Internet Connection Sharing (ICS) or tethering, though performance may not be as robust as in infrastructure-based networks.

Appendix A: OSI Stack Model

Layer	Name	Function
7	Application	Provides the interface between the user and network services
6	Presentation	Ensures data is in a readable format for the application layer
5	Session	Establishes, manages, and terminates sessions between applications
4	Transport	Provides reliable and ordered data transfer between processes
3	Network	Manages addressing, routing, and path determination
2	Data Link	Defines the protocol for transferring data across a physical link
1	Physical	Transmits raw bits over the communication medium

Tab. A1: OSI stack model layers and their functions.

Historical Remarks: The Open Systems Interconnection (OSI) model was developed by the International Organization for Standardization (ISO) in the late 1970s and early 1980s. The objective was to create a standardized framework for the development and implementation of network protocols to enable interoperability between different systems and vendors.

Although the OSI model has been largely replaced by the TCP/IP model in modern networking, it still serves as a useful conceptual tool to understand the various functions and layers involved in network communication. Many of the protocols defined in the OSI model remain relevant and are used in today's networks.

- **Layered Architecture:** The OSI model consists of seven layers, each with a distinct set of responsibilities. This layered architecture allows for separation of concerns, making it easier to design, develop, and troubleshoot network protocols.
- **Modularity:** The OSI model's modular design allows for the development and improvement of individual layers without impacting the entire system. This flexibility enables networks to evolve and adapt to new technologies and requirements.
- **Interoperability:** The OSI model promotes the development of interoperable network protocols by providing a common framework for their design. By adhering to the OSI model, different vendors and systems can communicate seamlessly, fostering a more open and interconnected networking environment.

Appendix B: Universal Resource Locator (URL)

URLs, or Universal Resource Locators, are a type of Uniform Resource Identifier (URI) that provide a means of locating and accessing resources on the World Wide Web. A URL is composed of different components that describe the resource's location and the method to access it.

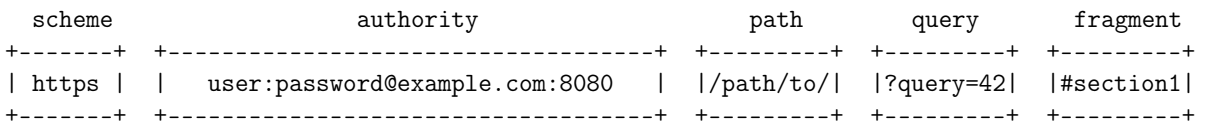


Fig. B1: URL structure and its components.

The primary components of a URL include:

- **Scheme:** The scheme specifies the protocol used to access the resource. Common schemes include HTTP (Hypertext Transfer Protocol), HTTPS (HTTP Secure), FTP (File Transfer Protocol), and mailto (for email addresses).
- **Authority:** The authority component is often composed of a username and password (optional) for authentication purposes, followed by the domain name (or IP address) of the server hosting the resource.
- **Port:** The port number, if specified, is used to connect to the server. If not provided, the default port for the specified scheme is used, e.g., 80 for HTTP or 443 for HTTPS.
- **Path:** The path represents the hierarchical structure of the resource on the server. It often corresponds to the directory and file structure of the server's filesystem.
- **Query:** The query component, if present, contains a series of key-value pairs separated by the '&' symbol, which provides additional information for processing the resource request.
- **Fragment:** The fragment identifier, if specified, refers to a specific part or section within the resource, often used for navigation purposes.

Appendix C: Reserved Ports

Port Number	Protocol	Usage
20	FTP	Data transfer
21	FTP	Control (command)
22	SSH	Secure Shell (remote login)
23	Telnet	Remote login
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name System
67	DHCP	Dynamic Host Configuration Protocol (server)
68	DHCP	Dynamic Host Configuration Protocol (client)
80	HTTP	Hypertext Transfer Protocol
110	POP3	Post Office Protocol 3
143	IMAP	Internet Message Access Protocol
443	HTTPS	HTTP Secure

Tab. C1: Reserved socket ports and their protocolar usage.

Note: In UNIX-based systems, port numbers below 1024 are considered "privileged" or "reserved" ports. These ports are locked for use by the operating system and system services, which often run with superuser (root) privileges. The main reason behind this restriction is security.

- **Root privileges for binding:** Lower-numbered ports are associated with essential system services (like SSH, FTP, and HTTP), they should be reserved for trusted applications and services that are authorized by the system administrator. By restricting access to these ports to only applications running with root privileges, the system minimizes the risk of unauthorized or rogue applications hijacking these critical ports, which could lead to security vulnerabilities or service disruption.
- **Least privilege principle:** After binding the port, the application may drop its root privileges to run as a less privileged user to reduce the potential impact of security vulnerabilities.
- **Ephemeral/unprivileged ports:** Ports with numbers above 1024 are available for non-privileged services and user applications, which can bind without root privileges.

Appendix D: TCP Header Flags and Optional Fields

Flag	Name	Usage
URG	Urgent	Indicates that the Urgent Pointer field is significant
ACK	Acknowledge	Indicates that the Acknowledgment Number field is significant
PSH	Push	Instructs the receiving host to immediately pass the data to the application
RST	Reset	Resets the connection due to an error or an unreachable host
SYN	Synchronize	Used for establishing connections; synchronizes sequence numbers
FIN	Finish	Indicates that the sender has finished transmitting data

Tab. D1: TCP header flags and their usage.

Option	Name	Usage
MSS	Maximum Segment Size	Specifies the maximum segment size that the sender is willing to receive
WSOPT	Window Scale	Allows window sizes larger than 65,535 bytes by defining a scale factor
SACK	Selective Acknowledgment	Acknowledges non-contiguous blocks of data and reports gaps
TSopt	Timestamp	Used for round-trip time measurement and PAWS (Protect Against Wrapped Sequence numbers)

Tab. D2: TCP optional fields and their usage.

- **Variable-length options:** Optional fields are variable in length and provide additional features, such as improved performance or error handling.
- **Negotiation during connection establishment:** Certain options, like MSS and WSOPT, are negotiated during the three-way handshake process.

Appendix E: IP Header Field - Transport-Layer Protocol

Protocol Number	Protocol	Usage
1	**ICMP**	Internet Control Message Protocol
2	IGMP	Internet Group Management Protocol
6	**TCP**	Transmission Control Protocol
8	EGP	Exterior Gateway Protocol
17	**UDP**	User Datagram Protocol
27	RDP	Reliable Datagram Protocol
41	**IPv6**	IPv6 encapsulation
47	GRE	Generic Routing Encapsulation
50	ESP	Encapsulating Security Payload
51	AH	Authentication Header
57	SKIP	Simple Key Management for IP
89	**OSPF**	Open Shortest Path First
97	EtherIP	Ethernet within IP Encapsulation
112	VRRP	Virtual Router Redundancy Protocol
115	L2TP	Layer 2 Tunneling Protocol
118	STP	Simple Transport Protocol
132	SCTP	Stream Control Transmission Protocol

Tab. E1: Expanded IP header field transport-layer protocol for the payload.

Note: The IP header contains a field named "Protocol", which is 8 bits long. This field is used to identify the transport-layer protocol carried in the IP datagram's payload. The protocol numbers are assigned by the **Internet Assigned Numbers Authority (IANA)** to standardize and maintain consistency in the identification of different transport-layer protocols.

- The IP protocol number helps routers and end systems identify the appropriate transport-layer protocol and process the IP datagram's payload accordingly. It ensures the correct protocol is used to handle the payload and maintain communication between the source and destination systems.
- **Future protocols:** The 8-bit field for protocol identification allows for up to 256 different protocol numbers. While not all of these numbers are currently assigned, this allows for the development and assignment of new transport-layer protocols in the future.

Referências

[1] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 8th edition, 2021. Chapters 1, 2, 3, 4, 5 & 6.

[2] S. J. Donahoo and Kenneth L. Calvert. *TCP/IP Sockets in C: Practical Guide for Programmers*. Morgan Kaufmann Publishers Inc., 2nd edition, 2009.

[3] João L. C. C. G. Sobrinho. "Acetatos de Redes de Computadores e Internet", 2022-2023.

[4] W. Richard Stevens. *UNIX Network Programming: Networking APIs: Sockets and XTI (Volume 1)*. Prentice Hall, 2nd edition, 1998.