

A Internet e Seus Conceitos Básicos

1. A Internet Atual:

- Conecta bilhões de usuários, dispositivos e enlaces de comunicação.
- Suporta uma grande variedade de dispositivos como notebooks, smart TVs, smartphones, sensores, webcams, consoles de jogos e utensílios domésticos.

2. Aplicações de Redes:

- Executadas em diversos dispositivos, essas aplicações transformam a forma de trabalhar, fazer comércio, interagir socialmente e se divertir.

3. Importância da Internet:

- A internet é essencial no cotidiano e, para quem estuda computação, vai além do simples uso.
- É fundamental entender os conceitos de organização das redes de computadores e da internet para sua utilização no presente e no futuro.

Conceitos básicos de Redes e Internet

1. **Redes de Computadores:** Conjunto de módulos processadores interligados por sistemas de comunicação, permitindo troca de informações e compartilhamento de recursos.

2. **Internet:** Rede global que interconecta bilhões de dispositivos utilizando os protocolos TCP e IP. Vai além de redes interligadas, funcionando como uma infraestrutura complexa de hardware e software que possibilita a comunicação global.

3. Partes principais da Internet:

- **Sistemas Finais:** Dispositivos de usuários como smartphones e computadores.
- **Núcleo da Rede:** Componentes que gerenciam a transferência de dados.
- **Redes de Acesso:** Conexões que ligam os sistemas finais ao núcleo da rede.

4. Serviços da Internet: Fornecem funcionalidades para:

- Roteamento (definir caminhos de dados).
- Correção de erros durante a transmissão.
- Controle de sobrecarga de componentes.

Esses serviços simplificam o desenvolvimento de aplicações, permitindo que os desenvolvedores foquem nas funcionalidades das aplicações, enquanto a infraestrutura da internet cuida do transporte seguro dos dados até o destino.

Entorno da Internet e Borda da Rede

1. Sistemas Finais (Hosts/hospedeiros):

- Dispositivos conectados à internet na periferia da rede.
- Executam aplicações de rede.

2. Categorias de Sistemas Finais:

- **Clientes:** Incluem desktops, notebooks, smartphones e tablets, geralmente de uso pessoal.
- **Servidores:** Máquinas mais robustas que armazenam e distribuem dados (páginas web, vídeos, e-mails). A definição de servidor depende do software que executa, não do hardware.

3. Função dos Servidores:

- Servidores fornecem serviços e respondem a requisições dos clientes, desempenhando papéis essenciais na infraestrutura de rede.

Redes de Acesso

1. Definição:

- Rede de acesso é o meio físico/enlace que conecta sistemas finais ao primeiro roteador(roteador de borda), ligando-os ao núcleo da rede.

2. Tipos de Meios Físicos:

- **Guiados:** Redes com fio, como cabo de fibra ótica(sinais luminosos), par de cobre trançado e cabo coaxial(sinais elétricos).
- **Não Guiados:** Redes wireless que utilizam sinais eletromagnéticos (ex.: redes domésticas sem fio, telefonia celular, satélite).

3. Categorias de Redes de Acesso:

- **Redes Residenciais:**
 - **DSL(linha digital de assinante):** Usa a linha telefônica com modem DSL.
 - **Cabo Coaxial:** Usa a infraestrutura de TV a cabo.
 - **FTTH (Fiber to the Home):** Conexão de fibra ótica direto para a residência(não é um padrão ou protocolo em si, mas apenas um conceito indicando que a fibra ótica chega até a residência), (uma grande fibra dividida em fibras individuais ao se aproximar das casas).
 - **Satélite:** Opção em áreas rurais, com menor velocidade.
 - **Discado:** Conexão antiga via linha telefônica, com velocidade baixa.
- **Redes Institucionais:**
 - **LANs(Local area network):** Usadas em ambientes corporativos, universidades e residências, geralmente com padrão **ethernet (cabos metálicos)** ou **wi-fi (IEEE 802.11)**.
 - **Redes de Telefonia Celular:** Acesso pela rede 3G/4G e, mais recentemente, 5G, oferecendo alta velocidade e grande alcance.

O Núcleo da Rede

1. Definição e Função:

- O núcleo da rede é formado por dispositivos (roteadores, switches) e enlaces de alta velocidade, interligando sistemas finais e ISPs, criando uma rede de redes para a interconexão global.

2. ISPs e Hierarquia:

- Existem milhares de ISPs com diferentes portes:

- **ISPs locais** conectam-se a ISPs regionais.
- **ISPs regionais** conectam-se a ISPs de nível 1, que têm alcance nacional/internacional.
- **ISPs de trânsito/globais** interligam outros ISPs, sem oferecer conexão direta ao usuário e geralmente administram cabos submarinos.

3. Ponto de Presença (PoP):

- Grupo de um ou mais roteadores (no mesmo local) na rede do provedor, onde os ISPs clientes podem se conectar para poderem acessar outras redes.

4. Multi-homing e Emparelhamento:

- **Multi-homing(exceto ISP nível 1):** ISPs podem se conectar a múltiplos provedores para garantir redundância.
- **Emparelhamento:** ISPs do mesmo nível podem conectar-se diretamente para reduzir custos, sem taxas entre eles.
- **Internet Exchange Point (IXP):** Estrutura independente onde diversos ISPs podem se conectar diretamente. No Brasil, o IX.br serve como ponto de troca de tráfego entre provedores.
- Um ISP cliente paga a um ISP provedor conforme a quantidade de tráfego que ele troca com o provedor.

A Rede como Serviço

- **Internet como infraestrutura:** A internet é uma base para prover serviços a várias aplicações, como e-mail, navegação, redes sociais, VoIP (voz sobre IP), P2P, e jogos distribuídos.
- **Aplicações distribuídas:** Executadas nos sistemas finais (hosts), e não nos comutadores do núcleo da rede.
- **Desenvolvimento de Aplicações:** Para criar esses programas, é necessário conhecimento de linguagens como Java, C ou Python.
- **Interface de Programação de Aplicação (API):** A API permite enviar dados pela rede sem se preocupar com a entrega, erros ou conversão dos bits em sinais.
- **Infraestrutura de Rede:** Gerencia todos os processos de envio de dados, passando pelo sistema operacional e dispositivos como roteadores e switches.
- **API Socket:** Será introduzida para comunicação de dados entre sistemas finais.

Atraso em Redes de Comutação de Pacotes

A internet idealmente transferiria dados entre sistemas finais de forma instantânea e sem perdas, mas restrições de rede causam atrasos, limitações na taxa de transferência e perdas de pacotes.

Parâmetros para Avaliação de Redes

- **Atraso, perda e vazão** são parâmetros fundamentais para diagnosticar e propor melhorias nas redes de computadores.

Tipos de Atraso

1. **Atraso de Processamento:** Tempo para o roteador analisar o cabeçalho do pacote e definir sua saída.
2. **Atraso de Fila:** Tempo de espera de um pacote para ser transmitido. Varia conforme o tráfego; alta demanda aumenta o tempo de fila.
3. **Atraso de Transmissão:** Tempo necessário para empurrar todos os bits do pacote no enlace. Depende do tamanho do pacote e da taxa de transmissão do enlace, mas não da distância.
4. **Atraso de Propagação:** Tempo para um bit atravessar o enlace até o próximo nó, dependendo do meio físico (fibra ótica, fios de cobre) e da distância entre roteadores.
5. **Atraso de processamento, atraso de fila, atraso de transmissão, atraso de propagação = atraso total.**

Efeitos dos Atrasos em Aplicações

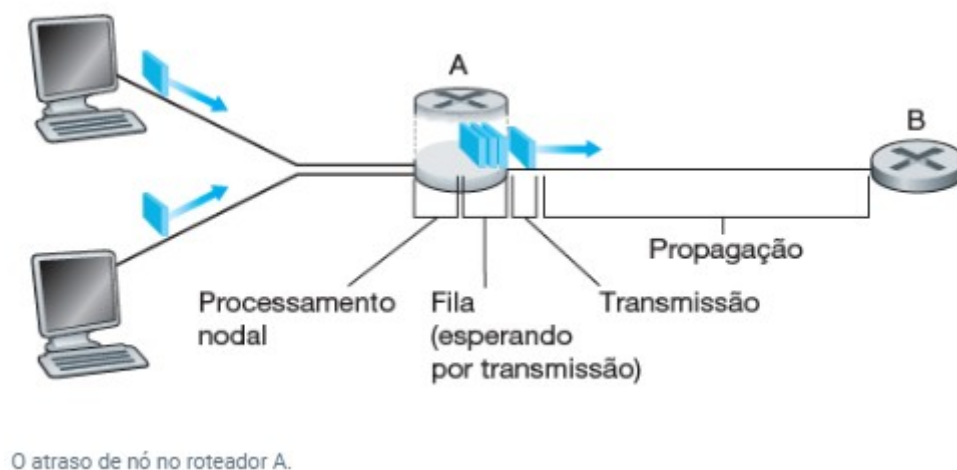
- **Aplicações em Tempo Real** (jogos interativos, videoconferências) são afetadas pelo atraso de propagação.
- **Transferências de Grandes Volumes de Dados** (ex.: arquivos) são mais impactadas pelo atraso de transmissão.

Variação no Atraso de Fila (**Jitter**)

- O atraso de fila pode variar entre pacotes, causando jitter, que afeta negativamente o streaming de multimídia, pois os pacotes chegam com intervalos irregulares, prejudicando a reprodução contínua e regular.

*Se o enlace estiver ocupado/com pacotes à espera, o recém-chegado entrará na fila (buffer, ou memória, do roteador).

*Pacote → sistema final de origem → vários roteadores → sistema final de destino.



Parâmetros de Redes e seu Impacto nas Aplicações

Perda de Pacotes

A perda de pacotes ocorre quando o buffer (fila) de um roteador atinge sua capacidade máxima e novos pacotes que chegam precisam ser descartados. Isso geralmente acontece quando:

- O tráfego está muito próximo da capacidade do enlace, gerando filas de espera.
- A intensidade de tráfego excede a capacidade de transmissão, fazendo com que a fila se acumule.
- Um sistema final considera que o fenômeno da perda é um pacote que foi transmitido para o núcleo da rede, sem nunca ter emergido dele no destino.
- O desenvolvedor da aplicação não precisa implementar uma técnica de controle de perdas, basta utilizar uma API (socket) correta. (infraestrutura de serviços)

Aplicações têm diferentes sensibilidades à perda:

- **Aplicações Críticas (e.g., transferência de arquivos):** Pacotes perdidos precisam ser retransmitidos, pois a perda impacta o funcionamento. Neste caso, a rede fornece mecanismos para corrigir as perdas.
- **Aplicações Tolerantes (e.g., streaming de vídeo):** Perdas de pacotes não afetam significativamente a experiência do usuário, pois a ausência de alguns pixels não é perceptível.

Atraso Fim a Fim

O atraso fim a fim é a soma dos atrasos de processamento, fila, transmissão e propagação ao longo de todos os nós (roteadores) entre a origem e o destino. Esse atraso é influenciado por:

- **Congestionamento:** Atrasos de fila altos indicam congestionamento, impactando especialmente as aplicações sensíveis ao atraso, como videoconferências.

A ferramenta **Traceroute (Tracert, no Windows)** permite medir o tempo de ida e volta entre cada nó, monitorando o atraso para análise de desempenho e identificação de pontos de congestionamento.

1	79 ms	2 ms	1 ms	192.168.0.1
2	13 ms	9 ms	8 ms	10.50.192.1
3	12 ms	10 ms	12 ms	201.17.8.85
4	15 ms	12 ms	11 ms	189.53.226.5
5	17 ms	20 ms	16 ms	200.244.19.75
6	20 ms	16 ms	11 ms	200.230.252.157
7	18 ms	10 ms	13 ms	200.244.18.22
8	119 ms	11 ms	13 ms	200.211.219.218
9	16 ms	12 ms	16 ms	200.143.253.22
10	11 ms	11 ms	11 ms	200.143.240.53
11	12 ms	126 ms	14 ms	200.143.240.65
12	28 ms	12 ms	14 ms	200.20.96.74
13	13 ms	14 ms	12 ms	200.20.96.82

Vazão Fim a Fim

A vazão é a taxa de bits por segundo que um hospedeiro destino recebe durante uma transferência. Existem duas métricas de vazão:

- **Vazão Instantânea:** Taxa momentânea de recepção de dados.
- **Vazão Média:** Taxa média da transferência ao longo de um período.

A vazão ideal depende do tipo de aplicação:

- **Telefonia IP e streaming:** Requerem baixa latência e uma vazão instantânea mínima constante.
- **Transferência de arquivos:** Beneficia-se de uma vazão média alta, enquanto o atraso não é uma prioridade.
- Se o arquivo consistir em F bits e a transferência levar T segundos para o hospedeiro B receber todos os F bits, a vazão média da transferência do arquivo é F/T bits/s.

A **vazão depende** das taxas de transmissão dos enlaces e do tráfego de outros sistemas na rede, podendo ser limitada por enlaces congestionados, como cabos submarinos, que servem múltiplos usuários simultaneamente.

*Se a taxa de chegada do pacote a um roteador exceder a sua capacidade de processamento por unidade de tempo, implicará um aumento do tamanho da fila, ou buffer. Uma vez que o tamanho máximo seja ocupado por pacotes, o comportamento do roteador será o de descartar o pacote; isto é, ele será perdido.

Arquitetura de Camadas

A **arquitetura de camadas** é uma maneira de organizar os protocolos de rede, hardware e software em uma rede de computadores, facilitando o desenvolvimento e a evolução da comunicação entre sistemas remotos.

O que é um Protocolo?

Um **protocolo de rede** define as regras e formatos das mensagens trocadas entre as entidades de rede, bem como as ações tomadas ao enviar ou receber essas mensagens. Ele é similar a um **protocolo humano**, como quando duas pessoas se cumprimentam antes de iniciar uma conversa. Por exemplo:

- Primeiro, dizemos "oi" e esperamos a resposta.
- Quando a resposta é dada, podemos seguir com a conversa (como perguntar as horas).

Da mesma forma, as redes de computadores dependem de protocolos para que as entidades se comuniquem corretamente. Para que a comunicação aconteça, os componentes da rede devem usar os mesmos protocolos.

Organização em Camadas

Para organizar os protocolos de rede e os sistemas que os utilizam, a arquitetura de redes foi dividida em **camadas**:

- **Cada camada executa um protocolo específico.**
- **Cada camada oferece serviços à camada superior** e utiliza os serviços da camada inferior.

O **ponto de ligação entre as camadas** é chamado de **interface**. Essas interfaces permitem a comunicação e a troca de informações entre as camadas, garantindo que a rede funcione de maneira estruturada.

Benefícios da Arquitetura de Camadas

A organização em camadas facilita a **modularidade** e **flexibilidade**, permitindo que alterações em uma camada não afetem as outras camadas. Além disso, ela torna mais fácil o desenvolvimento e a manutenção de sistemas complexos de comunicação. A camada mais baixa lida com a transmissão física de dados, enquanto camadas superiores tratam de tarefas mais abstratas, como controle de fluxo, confiabilidade, e aplicação dos dados.

Modelo de Referência OSI (RM-OSI)

O **modelo de referência OSI (Open Systems Interconnection)** foi proposto pela **Organização Internacional para Padronização (ISO)** no final dos anos 1970. Esse modelo organiza a comunicação em redes de computadores em **sete camadas** distintas, com cada camada responsável por um serviço específico. O modelo serve como uma referência para entender como os sistemas de rede devem funcionar, embora não seja utilizado diretamente em implementações, pois não especifica protocolos, mas sim os **serviços** que cada camada oferece.

As 7 Camadas do Modelo OSI:

1. Camada de Aplicação

- **Função:** Abriga as aplicações de rede que oferecem serviços, como transferência de arquivos e envio de mensagens.
- **Exemplo:** Protocolos como HTTP, FTP, SMTP.
- **Objetivo:** Permitir que as aplicações troquem informações entre os sistemas finais.

2. Camada de Apresentação

- **Função:** Trata da representação dos dados para que sistemas diferentes possam compreender as informações trocadas, como um "tradutor" de dados.
- **Serviços:** Compressão, criptografia e codificação de dados.
- **Objetivo:** Garantir que os dados sejam interpretados corretamente por sistemas diferentes.

3. Camada de Sessão

- **Função:** Gerencia a troca de dados entre sistemas e sincroniza essa troca.
- **Exemplo:** Se a conexão cair durante um download, a camada de sessão garante que o processo seja retomado a partir do ponto onde parou.
- **Objetivo:** Delimitar e sincronizar as sessões de comunicação.

4. Camada de Transporte

- **Função:** Transporta mensagens entre sistemas finais, garantindo que os dados cheguem corretamente, sem perdas e na ordem certa.
- **Exemplo:** Protocolos como TCP e UDP.
- **Objetivo:** Garantir uma troca de dados confiável e sem sobrecarregar a rede ou os sistemas.
- **Pacote:** Um pacote da camada de transporte é chamado **segmento**.

5. Camada de Rede

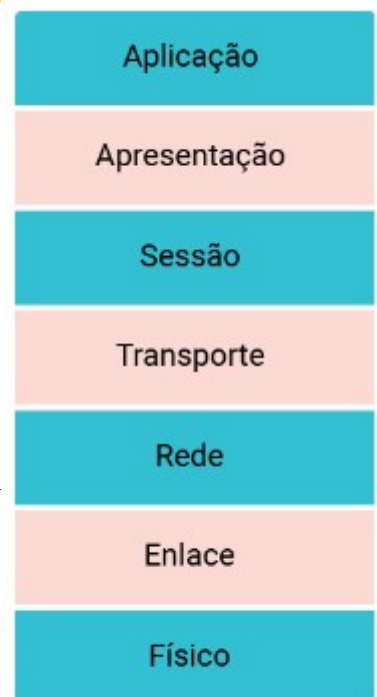
- **Função:** Responsável por determinar o caminho que os dados devem seguir de um sistema para outro.
- **Exemplo:** Protocolos como IP.
- **Objetivo:** Realizar o roteamento dos dados através da rede e garantir que os pacotes cheguem ao destino correto.
- **Pacote:** Um pacote da camada de transporte é chamado **datagrama**.

6. Camada de Enlace

- **Função:** Transfere os pacotes entre os nós da rede de maneira confiável, corrigindo erros que possam ocorrer durante a transmissão.
- **Exemplo:** Protocolos como Ethernet.
- **Objetivo:** Encaminhar pacotes de um nó ao próximo, garantindo a entrega sem erros.
- **Pacote:** Um pacote da camada de transporte é chamado **quadro**.

7. Camada Física

- **Função:** Lida com a transmissão física dos dados, convertendo os bits em sinais adequados para o meio de transmissão, como cabos de cobre ou fibra ótica.
- **Exemplo:** Tipos de cabeamento (Ethernet, fibra ótica) e sinais elétricos ou ópticos.
- **Objetivo:** Enviar os bits de dados como sinais físicos para o próximo nó da rede.



Resumo:

- **Objetivo principal do modelo OSI:** Definir uma estrutura padronizada para a comunicação entre sistemas de rede, separando funções em camadas específicas.
- **Cada camada tem funções distintas:** Desde a troca de dados com as aplicações, passando pela codificação, roteamento, controle de erros até a transmissão física dos bits.

Esse modelo é essencial para entender as redes de computadores e desenvolver sistemas de comunicação robustos e interoperáveis.

O modelo de referência OSI.

Arquitetura TCP/IP (ou Arquitetura Internet)

A **arquitetura TCP/IP** é a base para a comunicação na **Internet** e em muitas redes privadas. Ao contrário do modelo **OSI**, que é uma referência conceitual sem protocolos definidos, a arquitetura TCP/IP é implementada na prática e inclui um conjunto de protocolos específicos.

Modelos de Camadas

Originalmente, a arquitetura TCP/IP possui **quatro camadas: aplicação, transporte, inter-rede, e intrarrede**. No entanto, para fins didáticos, muitas vezes utiliza-se uma versão com **cinco camadas: aplicação, transporte, rede, enlace e física**. A camada **intrarrede é dividida em enlace e física**. Essa estrutura é chamada de **pilha de protocolos TCP/IP**, e cada camada tem uma função específica na comunicação de dados.

Diferença entre OSI e TCP/IP

- **Modelo OSI:** É um modelo de referência que não define protocolos, apenas descreve funções e serviços de cada camada.
- **Arquitetura TCP/IP:** É um conjunto de protocolos concretos que são usados para implementação de redes de comunicação, sendo amplamente adotado na internet.

Camadas na Arquitetura TCP/IP

1. Camada de Aplicação

- **Função:** Responsável por fornecer serviços de rede às aplicações. Ela inclui as funções de **apresentação** e **sessão**, que são tratadas no modelo OSI de maneira separada.
- **Protocolos Importantes:**
 - **HTTP** (HyperText Transfer Protocol): Para navegação web.
 - **DNS** (Domain Name System): Para resolver nomes de domínio.
 - **SMTP** (Simple Mail Transfer Protocol): Para envio de e-mails.
- **Exemplo:** Quando você acessa uma página web, o protocolo **HTTP** é usado para solicitar e transferir os dados da página.

2. Camada de Transporte

- **Função:** Garante que os dados trocados entre sistemas finais cheguem de forma confiável ou não, conforme o protocolo usado.
- **Protocolos:**
 - **TCP** (Transmission Control Protocol): Orientado à conexão, garantindo a entrega correta e em ordem dos dados. Ele realiza controle de fluxo e congestionamento.
 - **UDP** (User Datagram Protocol): Não orientado à conexão, mais rápido e sem garantias de entrega, adequado para aplicações que toleram perda de pacotes, como streaming de vídeo.
- **Exemplo:** Durante uma videoconferência, o **UDP** pode ser usado para reduzir o atraso, embora possa haver perdas de pacotes, enquanto uma transferência de arquivo usa **TCP** para garantir que todos os dados cheguem corretamente.

3. Camada de Rede

- **Função:** Responsável pelo endereçamento e roteamento de pacotes entre redes, utilizando o protocolo **IP**.
- **Protocolo Principal:**
 - **IP** (Internet Protocol): Define como os pacotes de dados são endereçados e roteados pela rede, garantindo que os dados cheguem ao destino.
- **Exemplo:** Quando você acessa um site, o endereço **IP** do servidor é usado para localizar o destino e rotear os pacotes até lá.

4. Camada de Enlace

- **Função:** Realiza a comunicação entre os nós (dispositivos) dentro de uma rede local, cuidando da transmissão de pacotes de dados de um nó para o outro.
- **Protocolos Usados:**
 - **Ethernet:** Usado em redes cabeadas.
 - **Wi-Fi:** Usado em redes sem fio.

- **Bluetooth:** Para comunicação de curto alcance entre dispositivos.
- **Exemplo:** Quando você se conecta a uma rede Wi-Fi, a camada de enlace gerencia a comunicação entre seu dispositivo e o roteador.

5. Camada Física

- **Função:** Cuida da transmissão dos dados em forma de sinais elétricos, ópticos ou de rádio, através dos meios físicos de comunicação.
- **Exemplo:** Em uma rede Ethernet, a camada física trata dos sinais elétricos transmitidos por cabos de cobre.

Resumo das Diferenças Principais:

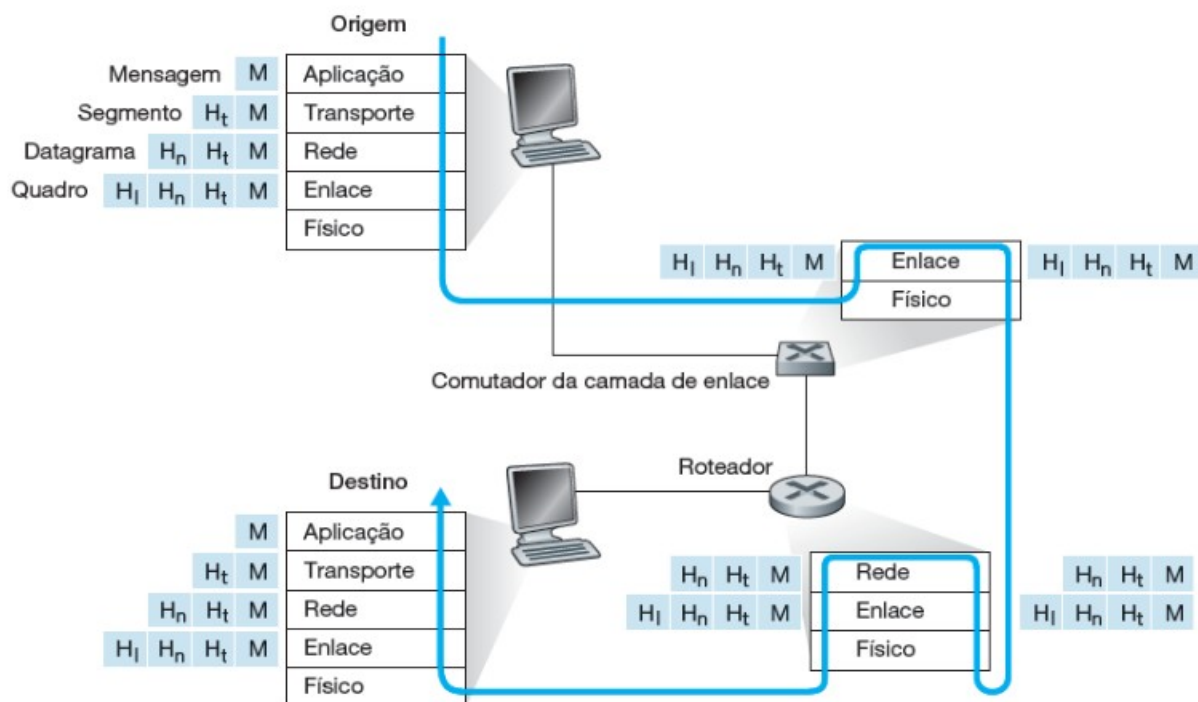
- **Modelo OSI:** Um modelo de referência teórico, sem protocolos definidos.
- **Arquitetura TCP/IP:** Um modelo prático, com protocolos reais implementados.

A arquitetura TCP/IP é fundamental para o funcionamento da internet moderna, conectando dispositivos de forma eficiente e com protocolos específicos para garantir a confiabilidade, a velocidade e a flexibilidade das redes.

Conceito de Encapsulamento

- **Definição:** Encapsulamento é o processo em que dados da camada de aplicação são passados para as camadas inferiores, com a adição de cabeçalhos específicos em cada camada.
- **Processo:**
 1. **Camada de Aplicação:** A mensagem gerada é passada para a camada de transporte.
 2. **Camada de Transporte:** Anexa seu cabeçalho (contendo informações de controle) à mensagem, formando a Unidade de Dados de Protocolo (PDU), que aqui é chamada de **segmento**.
 3. **Camada de Rede:** O segmento é passado para a camada de rede, que adiciona seu próprio cabeçalho (endereços de origem e destino), formando o **datagrama**.
 4. **Camada de Enlace:** Adiciona mais informações de cabeçalho, criando o **quadro** de enlace.
 5. **Camada Física:** Transmite os dados na forma de **bits** através do meio físico.
- **PDU (Unidade de Dados de Protocolo):** Cada camada tem um PDU composto por campos de cabeçalho e um campo de carga útil, que geralmente é um pacote da camada acima.
- **Desencapsulamento:** Quando os dados chegam ao destino, o processo inverso ocorre, com cada camada removendo o cabeçalho correspondente e reconstruindo os dados originais.
- **Caminho dos Dados:**
 1. **Sistema Final Emissor:** Dados percorrem as camadas de protocolo de baixo para cima.
 2. **Roteadores e Comutadores:** Atravessam as camadas de enlace no caminho de transmissão, mas não implementam todas as camadas da pilha de protocolos (por exemplo, roteadores executam apenas o protocolo IP na camada de rede).
 3. **Sistema Final Receptor:** Dados sobem pelas camadas de protocolo do receptor, onde o processo de desencapsulamento ocorre.

- **Observação:** Roteadores implementam protocolos até a camada 3 (rede), enquanto comutadores de enlace operam até a camada 2 (enlace), enquanto hospedeiros (como computadores e servidores) implementam todas as 5 camadas da pilha de protocolos.



Considere o processo de encapsulamento de dados de um sistema final A, que irá enviar pacote para o sistema final B. Qual é a sequência correta a respeito do processo de encapsulamento?

A camada de transporte passa o segmento à camada de rede, que adiciona informações de cabeçalho de camada de rede, criando um datagrama de camada de rede.

O encapsulamento sempre ocorre entre camadas adjacentes, no sentido "de cima para baixo" (considerando a sequência da mais alta para a mais baixa: aplicação, apresentação, sessão, transporte, rede, enlace, física).

*Comutação = troca, permuta, substituição, transformação, transmutação, metamorfose, alteração.

Resumo da História da Internet

Anos 1960:

- **Redes de telefonia:** Comutação de circuitos era a tecnologia dominante, usada para comunicação de voz.
- **Comutação de pacotes:** Invenção de uma alternativa eficiente à comutação de circuitos, permitindo interconectar computadores.
- **ARPAnet:** Primeira rede de comutação de pacotes, predecessora da internet pública.
- **Primeiro comutador de pacotes:** Instalado na UCLA em 1969.
- **Protocolo NCP(Network Control Protocol):** Primeiro protocolo de comunicação da ARPAnet.
- **Primeiro programa de e-mail:** Criado em 1972.

Anos 1970:

- **Interconexão de redes:** DARPA (Defense Advanced Research Projects Agency), com Vinton Cerf e Robert Kahn, cria a rede de redes.
- **Protocolos fundamentais:** TCP, UDP, e IP começam a ser desenvolvidos no final dos anos 1970.

Anos 1980:

- **TCP/IP:** Adotado oficialmente como padrão da ARPAnet em 1983, substituindo o NCP.
- **DNS:** Sistema de nomes de domínio é desenvolvido para facilitar a navegação na internet.

Anos 1990:

- **Desaparecimento da ARPAnet:** A ARPAnet dá lugar à internet pública.
- **World Wide Web:** Criada por Tim Berners-Lee (1989-1991), torna a internet acessível ao público.
- **Adoção comercial:** Surgem empresas como Amazon, eBay e Google, impulsionando o comércio eletrônico e a busca na web.
- **Crescimento exponencial:** Nos anos 90, empresas se tornam públicas e a internet se expande rapidamente, embora com uma bolha financeira entre 2000 e 2001.

Anos 2000 até o presente:

- **Banda larga:** Acesso à internet de alta velocidade, como cabo, DSL e FTTH, se populariza.
- **Aplicações de vídeo:** Surgem plataformas como YouTube, Netflix e Skype, permitindo comunicação e entretenimento online.
- **Wi-Fi e 4G/5G:** Aumento da conectividade móvel e a popularização de novas aplicações baseadas em localização.
- **Computação na nuvem:** Empresas de comércio e serviços começam a operar na nuvem (exemplo: Amazon EC2, Microsoft Azure).

*ARPA(Advanced Research Projects Agency).

História da Internet:

- **Década de 60:**
 - **1965:** Primeira comunicação entre dois computadores no MIT usando comutação de pacotes.
 - **1968:** BBN cria o Interface Message Processor (IMP), ganhando contrato com ARPANET.
 - **1969:** Primeira mensagem "LO" enviada na ARPANET, conexão entre universidades dos EUA.
- **Década de 70:**
 - **1972:** Introdução do e-mail e formação do grupo INWG para criar protocolos padrões.
 - **1973:** Conexão de ARPANET com a Inglaterra e Noruega, surgindo o termo "internet".
 - **1974:** Primeiros provedores comerciais de internet, com a ARPANET renomeada para Telenet. Surge o TCP.

- **1976:** A Rainha Elizabeth envia seu primeiro e-mail.
- **1979:** Uso de formulários USENET para notícias e discussões.
- **Década de 80:**
 - **1981:** NSF financia a CSNET para cientistas.
 - **1982:** TCP/IP torna-se o protocolo para ARPANET, definindo a estrutura da internet.
 - **1983:** Criação do DNS com domínios como .edu, .gov e .com.
 - **1984:** William Gibson usa o termo "ciberespaço".
 - **1985:** Primeiro domínio registrado: Symbolics.com.
 - **1986:** NSFNET conecta centros de supercomputadores a 56 Kbps, expandindo a internet.
 - **1989:** World.std.com se torna o primeiro provedor comercial de acesso discado.
- **Década de 90:**
 - **1990:** Tim Berners-Lee desenvolve o HTML.
 - **1991:** O CERN apresenta a World Wide Web.
 - **1992:** Primeiro áudio e vídeo transmitidos pela internet, popularizando "navegar".
 - **1993:** Marc Andreessen cria o navegador Mosaic. Internet chega a 2 milhões de computadores conectados.
 - **1994:** Netscape Communications nasce. Yahoo! é fundado.
 - **1995:** CompuServe, AOL e Prodigy oferecem acesso à internet. Amazon e eBay surgem.
 - **1996:** Guerra dos navegadores, com destaque para Microsoft e Netscape.
 - **1997:** Netflix é fundada. A Netscape anuncia que seu navegador será gratuito.
 - **1998:** Google é fundado, revolucionando as buscas. IPv6 é introduzido.
 - **1999:** AOL compra Netscape. Napster populariza o compartilhamento de arquivos.
- **Década de 2000-2010:**
 - **2000:** Estouro da bolha das pontocom. AOL se funde com Time Warner.
 - **2003:** Worm SQL Slammer se espalha rapidamente. WordPress é lançado.
 - **2004:** Facebook é lançado, marcando o início das redes sociais.
 - **2005:** YouTube e Reddit são fundados.
 - **2006:** Twitter é lançado; AOL muda seu modelo de negócios.
 - **2009:** Internet completa 40 anos.
- **Década de 2010-2020:**
 - **2010:** Facebook atinge 400 milhões de usuários. Instagram e Pinterest são lançados.
 - **2011:** Twitter e Facebook têm papel crucial nas revoltas do Oriente Médio.
 - **2012:** A Lei Stop Online Piracy é barrada, protegendo sites como YouTube.
 - **2013:** Edward Snowden revela programa de vigilância global da NSA.
 - **2015:** Instagram atinge 400 milhões de usuários, superando Twitter.
 - **2016:** Google lança o Google Assistant.
 - **2018:** Crescimento da Internet das Coisas, com sete bilhões de dispositivos.
 - **2019:** Lançamento das redes 5G.
- **2020-2022:**
 - **2021:** Mais de 4,66 bilhões de pessoas estão conectadas à internet.
 - **2022:** Starlink, da SpaceX, inicia serviços de internet via satélite.

*A especificação original do protocolo IP foi feita mais precisamente em setembro de 1981, na RFC 791.

*O programa de ciência de computadores na ARPA (Advanced Research Projects Agency), nos Estados Unidos, publicou um plano geral para a ARPAnet, a primeira rede de computadores por comutação de pacotes e uma ancestral direta da internet pública de hoje.

A internet oferece diversos serviços que impactam o comportamento humano, como governamentais, financeiros e educacionais.

- A falta de internet nos causa sensação de isolamento, evidenciada ao ativar o modo avião de um smartphone.
- Para o funcionamento desses serviços, dispositivos de rede precisam implementar funcionalidades complexas.
- Desde o início das redes de comunicação, foram desenvolvidas formas de otimizar a transmissão de dados, tornando o acesso rápido e eficiente.

Arquitetura em camadas: A internet, um conjunto de redes, depende de regras chamadas protocolos para uma comunicação segura, confiável e eficaz entre dispositivos. Dada a complexidade dessas regras, foi aplicada a técnica de “dividir para conquistar” organizando a rede em camadas.

- **Modelo em camadas:** Cada camada é responsável por regras específicas, numeradas da camada 1 (mais baixa) à camada n (mais alta). Cada camada oferece serviços para a superior e utiliza serviços da inferior.
- **Elementos das camadas:**
 - **Serviço:** Conjunto de funcionalidades que define o que a camada faz (ex. verificação de erros).
 - **Protocolo:** Implementa o serviço e executa as regras necessárias, como identificar computadores.
 - **Interface:** Ponto de comunicação entre camadas, permitindo a passagem de informações.
- **Implementação:** Protocolos e interfaces são implementados no hardware (ex. placa de rede) ou software (sistema operacional).



Essa organização facilita a comunicação entre dispositivos por meio de **Comunicação Vertical** (entre camadas) e **Comunicação Horizontal** (entre dispositivos da mesma camada em diferentes máquinas).

*Cada camada é responsável por uma determinada regra, ou seja, o serviço define o que a camada deve fazer, mas não como. A responsabilidade de implementar a regra, de definir o como, é do protocolo de rede implementado naquela camada. Portanto, é comum dizer que o protocolo é a implementação do serviço.

Comunicação Vertical e Horizontal nas Camadas de Rede:

- **Comunicação Vertical:** Refere-se ao fluxo de dados que, ao serem transmitidos, passam por todas as camadas do dispositivo de origem até a camada mais baixa, conectada ao meio de transmissão (como cabo ou ar). Esse fluxo desce camada por camada, com cada uma preparando o dado conforme suas regras e protocolos.
- **Processo na Origem:** O dado inicia na camada superior e desce até a camada 1, onde é enviado pelo meio de transmissão ao destino.
- **Processo no Destino:** O dado sobe pelas camadas, do nível mais baixo ao mais alto, até ser entregue ao destinatário final, passando pela mesma sequência de camadas em sentido inverso.
- **Comunicação Horizontal:** Ocorre entre camadas equivalentes de dispositivos diferentes. Exemplo: a camada 2 de origem, que verifica erros, prepara o dado para ser checado pela camada 2 do destino, garantindo a integridade da informação.



Encapsulamento e Comunicação entre Camadas na Rede:

- **Encapsulamento:** Processo em que cada camada adiciona um cabeçalho ao dado antes de transmiti-lo. Esse cabeçalho contém informações que serão lidas pela camada correspondente no dispositivo de destino, garantindo a comunicação horizontal entre camadas equivalentes.

- **Comunicação Horizontal Virtual:** A camada 2 de um dispositivo de origem adiciona um cabeçalho que só será lido pela camada 2 do dispositivo de destino, possibilitando a comunicação entre camadas de mesmo nível de forma virtual.
- **Processo Completo:** O dado passa por cada camada, que encapsula a informação adicionando seu cabeçalho até chegar à camada 1, que transmite ao destino. No destino, ocorre o processo inverso, com a remoção dos cabeçalhos até que o dado chegue ao usuário.
- **Unidade de Dados do Protocolo (PDU):** Em cada camada, a combinação do dado e seu cabeçalho cria uma PDU, essencial para que o protocolo de cada camada aplique as regras necessárias.
- **Vantagens e Desvantagens:** O encapsulamento aumenta o volume de dados transmitidos, mas compensa essa desvantagem com a modularidade, facilidade de manutenção e atualização dos protocolos, facilitando a evolução das redes.

*Ao realizar o encapsulamento, a unidade de dados do protocolo ou PDU (Protocol Data Unit, na sigla em inglês) é criada. A PDU é constituída pela informação que vem da camada superior (PDU da camada superior) e o cabeçalho da própria camada.



Modelo OSI

- **Origem e Propósito:** Desenvolvido pela ISO nos anos 1970, o modelo OSI (ISO/IEC 7498-1:1994/Open Systems Interconnection) é um modelo de referência em camadas para padronizar a comunicação entre diferentes sistemas de rede, sem a necessidade de modificar hardware ou software (Forouzan, 2010).
- **Estrutura e Funcionamento:**
 - O OSI é composto por **sete camadas**, cada uma com uma função específica. Embora o modelo descreva o que cada camada deve fazer, ele não determina os protocolos exatos a serem usados.
 - Ele é mais utilizado como referência para definir a função de cada camada, por isso, quando um protocolo é atribuído a uma camada específica, essa referência se baseia no modelo OSI.

- **Comunicação Vertical e Horizontal:**

- Na comunicação vertical, o dado flui de uma camada superior para uma inferior até ser transmitido pela camada física.
- A **comunicação horizontal** ocorre de maneira virtual entre camadas equivalentes em sistemas diferentes, facilitada pelos cabeçalhos adicionados em cada camada. Esses cabeçalhos formam a **Unidade de Dados de Protocolo (PDU)**.

- **Camadas do Modelo OSI:**

- **Camadas Superiores:** Incluem as camadas de **aplicação, apresentação e sessão**. Elas lidam com os serviços de rede voltados ao usuário e garantem a compatibilidade e a interoperabilidade entre diferentes sistemas.
- **Camadas Inferiores:** Englobam as camadas de **rede, enlace e física**, responsáveis pela transmissão de dados de um dispositivo para outro e pelas operações físicas de rede.
- **Camada de Transporte:** Atua como intermediária entre as camadas superiores e inferiores, organizando os dados das camadas inferiores para que estejam prontos para uso nas camadas superiores.

- **Encapsulamento de Dados:**

- Cada camada recebe os dados da camada superior, adiciona um cabeçalho próprio (encapsulamento) e passa para a camada seguinte. Esse processo repete-se até o dado atingir a camada física para transmissão.
- No destino, o processo é invertido, desencapsulando os dados à medida que sobem pelas camadas até chegar ao usuário.
- **Vantagens e Desvantagens:** Embora o encapsulamento aumente o tamanho dos dados transmitidos, ele proporciona vantagens como modularidade, facilidade de manutenção e atualização de protocolos, permitindo a evolução das redes.

O modelo OSI, embora pouco utilizado diretamente nos protocolos atuais, serve como uma estrutura de referência essencial para entender a comunicação em redes e o funcionamento das camadas.



Camadas de Aplicação, Apresentação e Sessão

Camada de Aplicação

- **Função principal:** Está mais próxima do usuário e oferece serviços como:
 - Serviço web
 - Correio eletrônico
 - Transferência de arquivos

- Streaming de áudio e vídeo
- Compartilhamento de arquivos
- **Execução:** Realizada por processos em dispositivos dos usuários.
- **Tarefas adicionais:** Traduzir, criptografar e comprimir dados.

Camada de Apresentação

- **Função principal:** Garante a interoperabilidade entre sistemas heterogêneos (diversos dispositivos e sistemas operacionais).
 - **Transformação dos dados:** A camada converte entre diferentes formatos e assegura a comunicação entre dispositivos distintos.
 - **Responsabilidades:**
 - Conversão de formatos
 - Compressão de dados
 - Criptografia
- **Objetivo:** Permitir que os dados trocados sejam compreensíveis independentemente do dispositivo ou sistema operacional em uso.

Camada de Sessão

- **Função principal:** Organiza e gerencia a comunicação entre dispositivos, permitindo o estabelecimento de sessões de comunicação.
 - **Serviços básicos:**
 - Controle de diálogo: Define quem pode transmitir em determinado momento.
 - Sincronização: Controla a transmissão (full duplex ou half duplex).
- **Objetivo:** Assegurar que a comunicação entre dispositivos seja organizada e eficiente, com controle de fluxo e erros.

Tarefas das Camadas

1. **Camada de Aplicação:** Tradução, criptografia e compressão de dados.
2. **Camada de Apresentação:** Garantir a interoperabilidade e transformação dos dados.
3. **Camada de Sessão:** Estabelecer, gerenciar e encerrar sessões, garantindo sincronização e controle do diálogo.

Essas camadas formam a base para garantir que as comunicações em rede sejam realizadas de forma eficiente, organizada e segura.

Camadas de Transporte e Rede

Camada de Transporte

- **Função principal:** Garantir a entrega de dados de processo a processo entre os usuários.
- **Responsabilidades:**
 - **Segmentação e remontagem:** Divide os dados em segmentos e garante que cheguem corretamente à camada de aplicação do destino.
 - **Controle de erros:** Verifica e corrige erros que possam ocorrer durante a comunicação entre os processos.
 - **Controle de fluxo:** Impede que o processo de origem sobrecarregue o processo de destino.

- **Controle de conexão:** Pode ser orientada ou não à conexão, gerenciando a criação e o encerramento de conexões.
- **Endereçamento do ponto de acesso:** Utiliza o endereço de porta para entregar os dados ao processo correto no destino.
- **Controle de congestionamento:** Monitora e lida com a sobrecarga de dispositivos intermediários durante a comunicação.
- **Objetivo:** Transferir pacotes da origem ao destino e garantir a comunicação entre os processos.

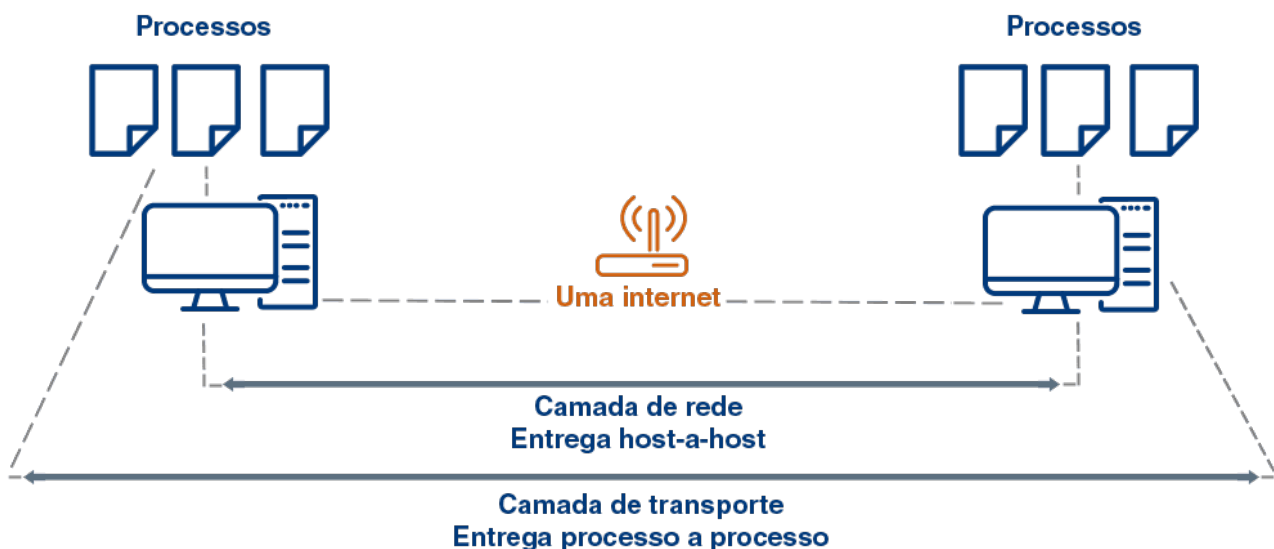
Camada de Rede

- **Função principal:** Determinar o caminho de origem a destino e realizar a comunicação máquina a máquina.
- **Responsabilidades:**
 - **Endereço lógico:** A camada de rede usa endereços lógicos para garantir que os dispositivos intermediários encaminhem os dados corretamente até o destino.
 - **Roteamento:** Estabelece o caminho entre a origem e o destino, com dispositivos intermediários encaminhando os dados para os dispositivos de destino.
- **Objetivo:** Organizar os bits em quadros e fornecer a entrega de dados entre máquinas, considerando dispositivos intermediários e roteamento.

Tarefas Principais:

1. **Camada de Transporte:** Gerencia a comunicação entre processos, garantindo que os dados cheguem corretamente.
2. **Camada de Rede:** Determina o caminho e realiza a comunicação entre máquinas, utilizando endereços lógicos e roteamento.

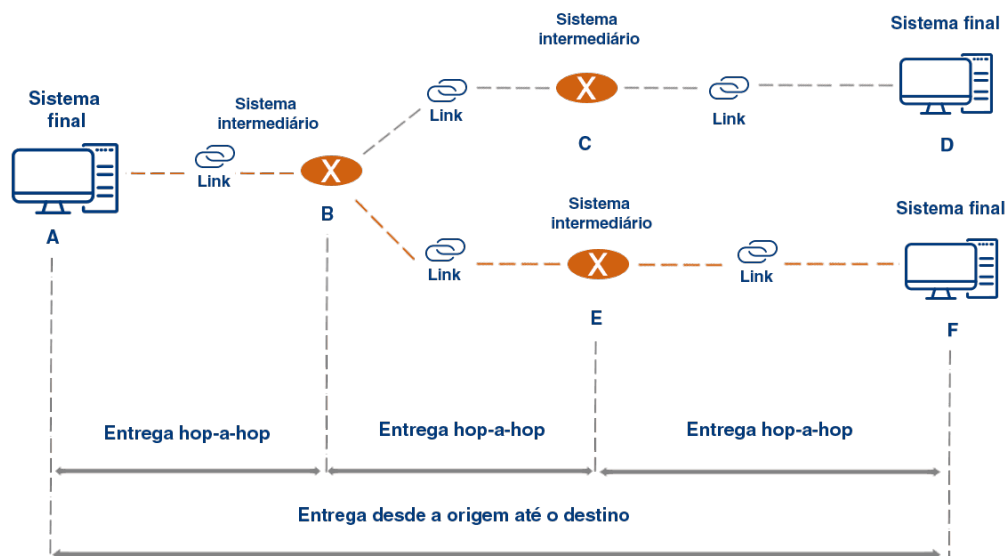
Essas camadas garantem uma comunicação eficiente entre os dispositivos na rede, com foco no controle de fluxo, erros, congestionamento e no encaminhamento correto dos dados.



Camadas de Enlace e Física

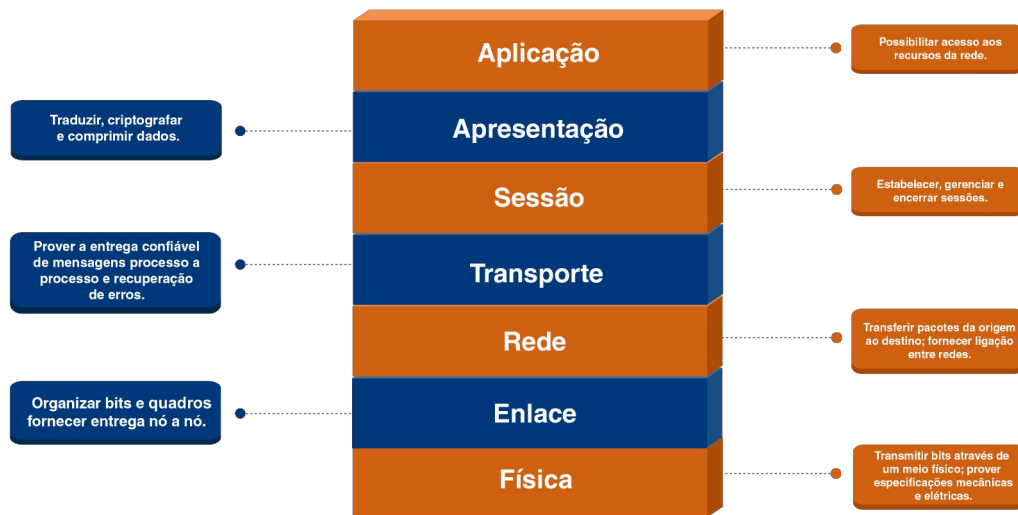
Camada de Enlace

- **Função principal:** Garantir a comunicação confiável entre dispositivos adjacentes, corrigindo erros e entregando dados à camada de rede de forma livre de falhas.
- **Responsabilidades:**
 - **Controle de erros:** Detecta e corrige erros durante a transmissão.
 - **Controle de acesso ao meio:** Garante que dispositivos compartilhando o mesmo meio de transmissão se comuniquem sem conflitos.
 - **Endereçamento físico:** Identifica o dispositivo de destino na rede através de um endereço físico, que varia conforme o caminho.
 - **Controle de fluxo:** Evita sobrecarga nos dispositivos de destino, regulando a taxa de transmissão.
 - **Enquadramento:** Encapsula os dados em quadros com delimitadores para marcar o início e fim dos pacotes, facilitando a identificação e processamento.



Camada Física

- **Função principal:** Transmitir os dados pelo meio físico de comunicação.
- **Responsabilidades:**
 - **Taxa de dados:** Define a velocidade de transmissão (ex: Mbps), controlando a duração de cada bit.
 - **Sincronização dos bits:** Garante que o transmissor e receptor operem na mesma taxa de bits, evitando falhas de sincronismo.
 - **Topologia física:** Define a estrutura de conexão entre os nós da rede, como ponto a ponto ou ponto-multiponto.
 - **Modo de transmissão:** Estabelece como os dados serão enviados:
 - **Simplex:** Dados em um único sentido.
 - **Half duplex:** Dados em dois sentidos, mas não simultaneamente.
 - **Full duplex:** Dados em dois sentidos simultaneamente.



Comparação entre as camadas de Enlace e Física:

- **Enlace:** Comunicação confiável entre máquinas adjacentes, com controle de erros, fluxo, e acesso ao meio.
- **Física:** Transmissão dos dados por meios físicos, com foco na codificação, taxa de dados, e topologia da rede.

Essas camadas trabalham juntas para garantir que os dados sejam corretamente transmitidos e recebidos pelos dispositivos na rede, desde a origem até o destino.

*As camadas de rede, enlace e física permitirão que os dados possam sair do dispositivo de origem e alcançar o dispositivo de destino. A camada física vai permitir que o fluxo de bits flua pelo meio de comunicação; a camada de enlace vai garantir que não houve erro na transmissão do fluxo de bits e, de acordo com o caminho definido pela camada de rede, encaminhará os dados para o próximo nó.

Arquitetura e Evolução do Protocolo TCP/IP

Protocolo TCP/IP

- Criado em 1974 por Cerf e Kahn, o TCP/IP foi desenvolvido para permitir a interconexão de diversas redes de comunicação. Tornou-se o padrão para a internet.

Arquitetura TCP/IP

- Composta por **quatro camadas**:
 1. **Aplicação:** Responsável pelos protocolos de comunicação de software.
 2. **Transporte:** Garante a entrega confiável dos dados entre máquinas.
 3. **Internet:** Gerencia a endereçamento e roteamento de pacotes de dados.
 4. **Acesso à Rede:** Lida com a transmissão de dados no meio físico.
- **Camadas alternadas na literatura:**

1. **Camada de Internet:** Também conhecida como **rede** ou **inter-rede**.
2. **Camada de Acesso à Rede:** Pode ser chamada de **enlace**, **host-rede**, **intranrede** ou **host-network**.

Funções Acumuladas

- A camada **Aplicação** do TCP/IP acumula as funções das camadas **Apresentação** e **Sessão** do modelo OSI.
- A camada **Acesso à Rede** executa as funções das camadas **Enlace** e **Física**.

Diferenças entre o Modelo OSI e o TCP/IP

- **Modelo OSI:** Baseado principalmente nas funcionalidades das camadas.
- **Arquitetura TCP/IP:** Vai além das funcionalidades e desenvolve protocolos independentes e hierárquicos, onde protocolos de nível superior são apoiados por protocolos de nível inferior.

Pilha de Protocolos TCP/IP

- A pilha de protocolos TCP/IP é o conjunto de todos os protocolos da arquitetura e foi desenvolvida pelo IETF (Internet Engineering Task Force), uma comunidade internacional que busca a evolução da arquitetura da internet.



Arquitetura TCP/IP



Modelo OSI



Camadas: Funções e Principais Protocolos

Camada de Aplicação

- Engloba os serviços das camadas **aplicação**, **apresentação** e **sessão** do modelo OSI.
- Proporciona acesso a serviços na internet, tanto para usuários quanto para a operação da rede.
- **Principais serviços e protocolos:**
 - **Web:** HTTP, HTTPS
 - **Correio Eletrônico:** SMTP, POP, IMAP
 - **Nomes:** DNS
 - **Transferência de Arquivos:** FTP, TFTP
 - **Áudio e Vídeo em Tempo Real:** RTP
 - **Configuração Automática de Estações:** DHCP
- **Arquiteturas:**
 - **Cliente-servidor:** Cliente requisita serviço do servidor.
 - **P2P (Peer-to-Peer):** Troca de informações direta entre usuários, sem servidores.
- **PDU (Unidade de Dados do Protocolo):** As mensagens trocadas entre processos.

Camada de Transporte

- Garante a entrega dos dados entre processos do usuário.

- **Principais protocolos:**
 - **TCP (Transmission Control Protocol):** Orientado à conexão, confiável, controla erros, congestionamento e fluxo. Garante a entrega na ordem correta.
 - **UDP (User Datagram Protocol):** Não orientado à conexão, sem confiabilidade, usado em aplicações que exigem baixa latência, como videoconferências.

Camada de Internet

- Permite que os dados sejam roteados da origem ao destino.
- **Principal protocolo: IP (Internet Protocol)** – Define endereços lógicos (IP) e roteia datagramas.
- **Versões:** IPv4 e IPv6.
- **Características:** Serviço de "melhor esforço" – Não garante entrega, ordem ou ausência de erro (corrigido por camadas superiores).
- **Protocolos de suporte:**
 - **ICMP:** Mensagens de controle e erro.
 - **ARP:** Tradução de endereços lógicos para físicos.
 - **IGMP:** Comunicação multicast para grupos de estações.

Camada de Acesso à Rede

- Não tem um protocolo específico definido pela arquitetura TCP/IP.
- Engloba os serviços das camadas de **enlace** e **física** do modelo OSI.
- **Padrões:** Utiliza os padrões IEEE 802, como **Ethernet** e **WiFi**.

Evolução dos Protocolos

- A evolução dos protocolos da camada de aplicação começou com serviços textuais simples (ex: Serviço Web), e atualmente suporta tráfego de vídeo, segurança e transações financeiras.
- **Exemplo:** O protocolo **HTTPS** evoluiu de um protocolo textual para binário para suportar sistemas complexos como CRMs, ERPs, e outros.

Correlação com o Modelo OSI

- A arquitetura TCP/IP, com suas camadas, pode ser correlacionada com o modelo OSI, mas evoluiu principalmente nos protocolos, especialmente da camada de aplicação.

4o mini

Você disse:

pode resumir de outra maneira?

O ChatGPT disse:

ChatGPT

Resumo: Camadas e Protocolos da Arquitetura TCP/IP

Camada de Aplicação

- Integra os serviços das camadas **aplicação**, **apresentação** e **sessão** do modelo OSI.
- Permite o acesso a serviços na internet, tanto para usuários quanto para suporte de rede.

- **Protocolos e serviços:**
 - **Web:** HTTP, HTTPS
 - **E-mail:** SMTP, POP, IMAP
 - **DNS:** Sistema de Nomes
 - **Transferência de Arquivos:** FTP, TFTP
 - **Multimídia:** RTP
 - **Configuração de Estações:** DHCP
- **Arquiteturas:**
 - **Cliente-servidor:** Cliente solicita serviço ao servidor.
 - **P2P (Peer-to-Peer):** Troca direta de dados entre usuários, sem servidor central.

Camada de Transporte

- Garante a entrega confiável ou não confiável dos dados entre processos.
- **Protocolos:**
 - **TCP:** Protocolo confiável, orientado à conexão, com controle de erros e fluxo.
 - **UDP:** Protocolo não confiável, sem garantia de entrega ou ordem, ideal para aplicações que exigem baixa latência (ex: videoconferências).

Camada de Internet

- Responsável pelo roteamento dos dados entre o remetente e o destinatário.
- **Principal protocolo: IP (Internet Protocol)**, com versões **IPv4** e **IPv6**.
- **Características:** Serviço de "melhor esforço", sem garantia de entrega, ordem ou correção de erros.
- **Protocolos auxiliares:**
 - **ICMP:** Comunicação de erros e controle.
 - **ARP:** Tradução de endereços lógicos (IP) para físicos (MAC).
 - **IGMP:** Suporte a comunicação multicast.



Camada de Acesso à Rede

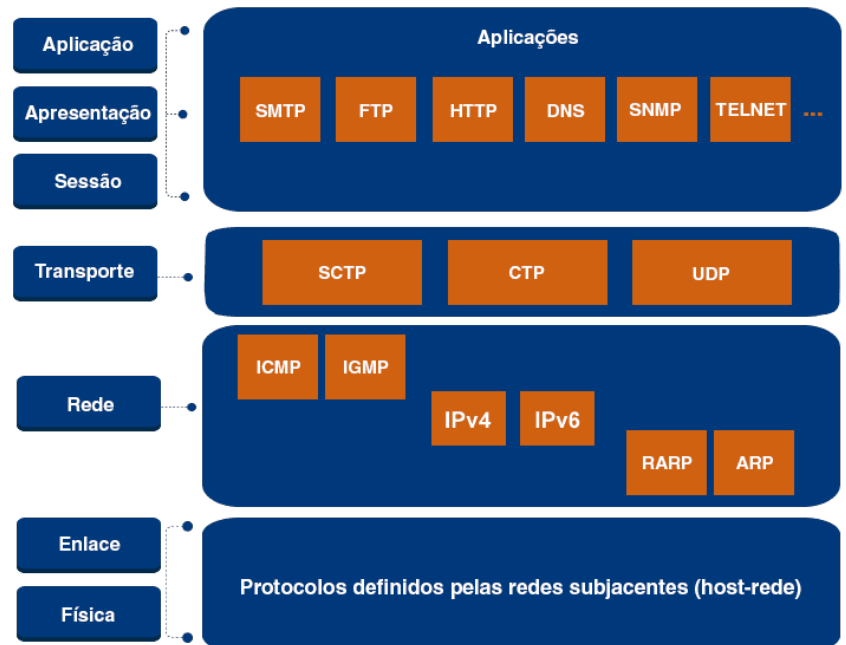
- Não definida explicitamente na arquitetura TCP/IP, mas abrange as camadas **enlace** e **física** do modelo OSI.
- Utiliza padrões como **Ethernet** e **WiFi** (IEEE 802).

Evolução dos Protocolos

- Inicialmente focada em serviços simples como o **HTTP** para páginas textuais, a camada de aplicação evoluiu para suportar multimídia, transações financeiras e segurança, com o **HTTPS** como exemplo de modernização.

Comparação com o Modelo OSI

- Embora a arquitetura TCP/IP seja mais simples, seus protocolos evoluíram para atender às necessidades de serviços modernos, refletindo um avanço em relação ao modelo OSI.



Crescimento das Redes e Camadas da Arquitetura TCP/IP

- **Crescimento das Redes:**
 - As redes de computadores cresceram significativamente nos últimos anos, impulsionadas pelos serviços oferecidos pela Internet, como compras online, pagamentos, streaming de filmes, jogos e redes sociais.
- **Camada de Aplicação:**
 - A camada de aplicação da arquitetura **TCP/IP** é responsável por implementar os serviços acessados via Internet ou Intranet.
 - Nessa camada, os processos que permitem o acesso a conteúdos, envio de e-mails, jogos online, redes sociais, entre outros, são executados.
 - A comunicação entre esses serviços ocorre no nível da aplicação, possibilitando a interação entre usuários e recursos na rede.
- **Camada de Transporte:**
 - A camada de transporte garante a **confiabilidade** na troca de mensagens entre os processos da camada de aplicação.
- **Importância para Profissionais de TI:**
 - É crucial que os profissionais de **Tecnologia da Informação** entendam as diferentes arquiteturas de aplicação e os serviços oferecidos.

- Devem conhecer como a camada de transporte assegura a comunicação confiável entre os dispositivos de origem e destino.

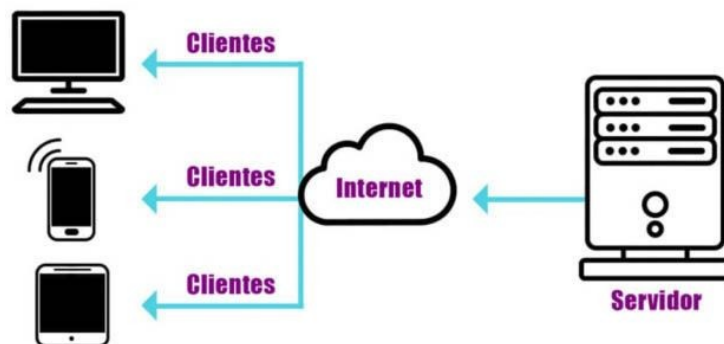
Camada de Aplicação

- **Importância das Redes:**
 - As redes de computadores são essenciais no cotidiano, permitindo interações e a realização de diversas tarefas.
- **Camada de Aplicação:**
 - A camada de aplicação é responsável por conectar os usuários à rede, proporcionando a execução de serviços como navegação na web, envio de e-mails e jogos online.
 - Ela é a camada de **mais alto nível** no modelo OSI e fornece a interface entre o usuário e os sistemas.
- **Exemplo Prático:**
 - Em uma transação de compra com cartão de crédito, o **software da máquina de cartão** que processa o pagamento opera na camada de aplicação. O software de aplicação é a interface que interage diretamente com os serviços da rede.
- **Exemplos de Softwares de Aplicação:**
 - Navegador web
 - Cliente de e-mail
 - Jogos executados em rede
- **Arquiteturas de Aplicações:**
 - Para desenvolver uma aplicação de rede, é necessário escolher uma **linguagem de programação** que tenha funções para comunicação em rede.
 - As arquiteturas mais comuns para aplicações em rede são:
 - **Cliente-servidor:** O cliente solicita serviços de um servidor.
 - **Peer-to-peer (P2P):** Usuários trocam informações diretamente entre si, sem a necessidade de um servidor central.

Arquitetura Cliente-Servidor

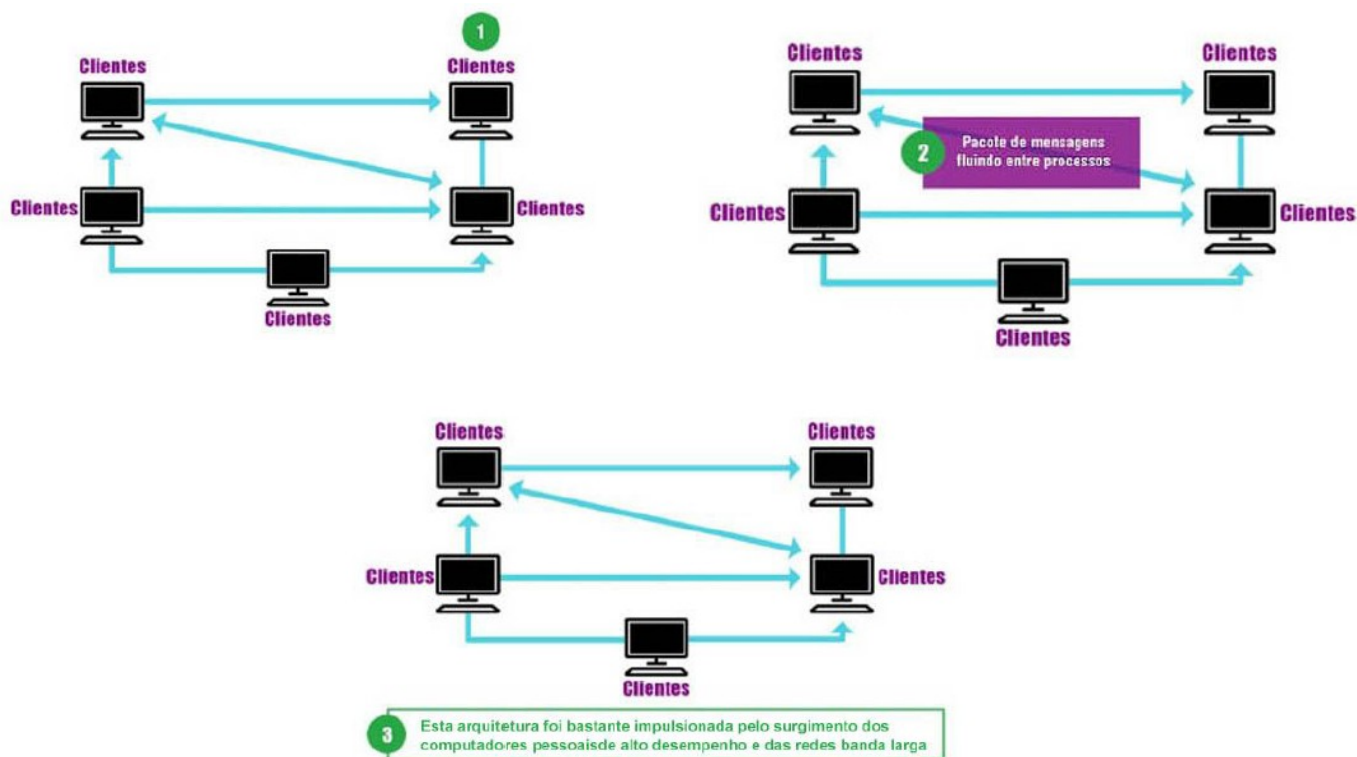
- **Definição:**
 - A arquitetura **cliente-servidor** envolve duas entidades principais: **cliente** e **servidor**.
 - O servidor executa operações e fica aguardando requisições dos clientes, que geralmente são vários, enquanto o servidor é único.
- **Funcionamento:**
 - O **cliente** envia uma mensagem ao servidor, especificando a operação a ser realizada e os dados necessários.
 - O **servidor** recebe a mensagem, processa a solicitação e envia a resposta ao cliente.
 - O servidor pode atender imediatamente, gerar um processo-filho, enfileirar a solicitação ou criar uma thread para o atendimento.
- **Exemplo Prático:**

- Em uma aplicação web, ao acessar um site de receitas, o **browser** do usuário (cliente) envia uma solicitação ao servidor, que processa a requisição e retorna a receita.
- **Função Cliente vs. Servidor:**
 - A distinção entre cliente e servidor depende da **função** desempenhada pelo software, não do tipo de equipamento.
 - Servidores têm alta confiabilidade, MTBF alto e recursos redundantes, sendo responsáveis pelo processamento e armazenamento de dados.
 - **Exceção:** um processo pode atuar simultaneamente como cliente e servidor.
- **Exemplo de Processos Múltiplos:**
 - Quando o servidor web (servidor) precisa de dados do banco de dados, ele se comunica com o servidor de banco de dados, atuando como **cliente** neste caso, para obter as informações necessárias.



Arquitetura Peer-to-Peer (P2P)

- **Definição:**
 - Na **arquitetura Peer-to-Peer (P2P)**, todos os processos envolvidos desempenham funções similares, ao contrário da arquitetura cliente-servidor, onde há distinção clara entre cliente e servidor.
- **Características:**
 - **Distribuição de Processamento e Armazenamento:** Em P2P, o processamento e o armazenamento de informações são distribuídos entre os hospedeiros (ou hosts), que são dispositivos como computadores, roteadores e smartphones.
 - **Escalabilidade:** Essa distribuição confere maior escalabilidade ao sistema, em comparação com a arquitetura cliente-servidor.
 - **Usuários Comuns:** Na P2P, os participantes são, geralmente, usuários comuns executando programas em seus dispositivos pessoais (desktops e notebooks).
- **Hospedeiro:**
 - O termo **hospedeiro (host)** se refere a qualquer equipamento conectado à rede capaz de trocar informações com outros dispositivos.
- **Modelo de Arquitetura:**
 - O fluxo de informações em P2P envolve a troca direta entre os hospedeiros, sem a necessidade de um servidor central.



*O que determina se a função desempenhada pela entidade é de cliente ou de servidor é seu comportamento definido pelo algoritmo executado, ou seja, pelo software.

*Na arquitetura peer-to-peer, todas as entidades desempenham a mesma função, possuindo igual capacidade de armazenamento das informações do sistema. Dessa forma, uma informação procurada pode estar com qualquer um dos participantes da rede, estando distribuída por todo o sistema.

Camada de Aplicação e Protocolos

- **Função da Camada de Aplicação:**
 - A camada de aplicação é onde os processos dos usuários são executados. Ela permite a interação com o sistema, mas para que diferentes aplicações troquem dados, é necessário um **protocolo de aplicação**.
- **Definição de Protocolo de Camada de Aplicação:**
 - Um **protocolo da camada de aplicação** define como os processos de uma aplicação, em sistemas finais diferentes, trocam mensagens. Ele especifica:
 - **Tipos de mensagens:** como requisição e resposta.
 - **Sintaxe das mensagens:** os campos da mensagem e como são delimitados.
 - **Semântica dos campos:** o significado das informações nos campos.
 - **Regras de envio e resposta de mensagens:** determina quando e como as mensagens são enviadas e respondidas.
- **Protocolo e Funcionamento Local:**

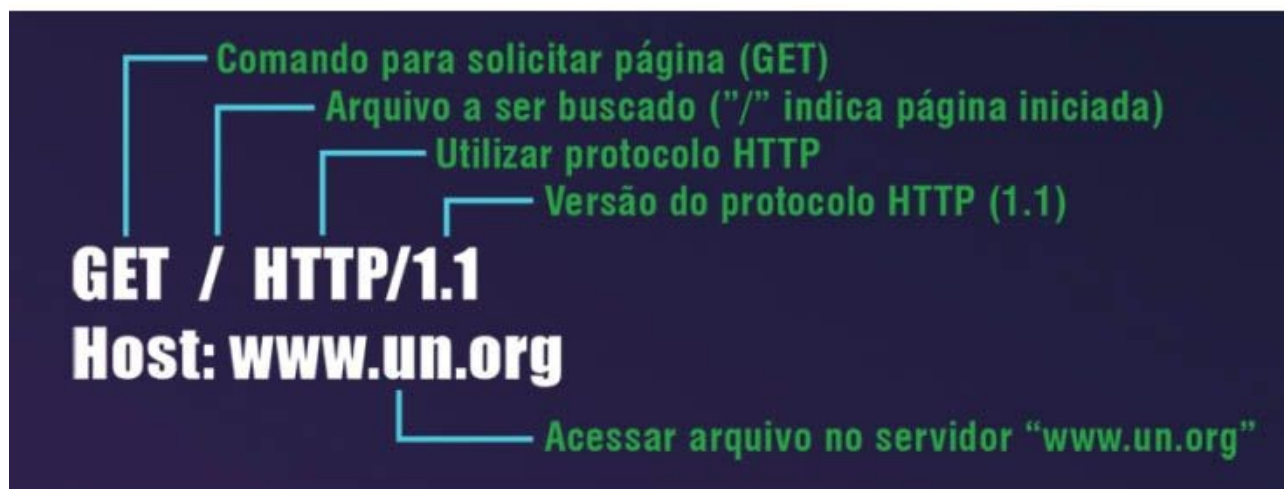
- O **algoritmo** da camada de aplicação determina o funcionamento no ambiente local, enquanto o **protocolo** estabelece como as aplicações em diferentes hospedeiros se comunicam.
- **Especificações de Protocolos:**
 - Protocolos públicos da internet são definidos por **RFCs (Request For Comments)**, que podem ser acessadas por qualquer pessoa para implementar softwares compatíveis.
- **Protocolos Importantes na Internet:**
 - **HTTP (HyperText Transfer Protocol):** usado em serviços web, é amplamente associado à navegação na internet.
 - **SMTP, POP3 e IMAP:** protocolos para serviço de correio eletrônico (e-mail).
 - **DNS (Domain Name System):** sistema de resolução de nomes, essencial para a tradução de nomes de domínio em endereços IP.

Esses protocolos são fundamentais para a comunicação e operação de serviços na internet.

Serviço Web e Protocolo HTTP

- **HTTP (Hypertext Transfer Protocol):**
 - Protocolo padrão para transferência de páginas web, definido pelas RFCs 1945 e 2616.
 - Criado em 1991 no CERN para facilitar a colaboração científica através de hipertextos.
 - Funciona em duas etapas essenciais:
- **Etapas 1:**
 - O usuário acessa uma página web digitando seu endereço no navegador ou clicando em um hiperlink.
 - O navegador e o servidor seguem um padrão que determina como a página é solicitada e transferida.
- **Etapas 2:**
 - Esse padrão é o protocolo HTTP, que opera sobre o protocolo TCP para transmitir as mensagens.
 - A interação começa com uma **requisição HTTP** enviada pelo cliente ao servidor, geralmente com:
 - **Texto ASCII** (formato de texto padrão),
 - O termo **GET**,
 - A página solicitada, o protocolo usado, e o servidor a ser contatado.
- **Conceitos Importantes:**
 - **ASCII:** Código binário para representar caracteres (128 símbolos como letras, sinais e controles).
 - **Hipertexto:** Documento com links que levam a outros documentos.

O protocolo HTTP estabelece a comunicação estruturada entre cliente e servidor, essencial para a exibição de páginas web.



Serviço de Correio Eletrônico (E-mail)

1. Origem e Evolução do E-mail

- **Primeiros Sistemas:** Projetados para troca simples de arquivos, onde o destinatário era indicado na primeira linha do texto.
- **ARPANET e Padronização:**
 - Em 1982, foram criadas as RFCs 821 e 822 para definir o protocolo de transmissão (SMTP) e o formato da mensagem.
 - Mensagens eram limitadas ao **código ASCII**, o que restringia o uso de alfabetos não-latinos, multimídia (áudio/vídeo) e caracteres acentuados.
- **Multipurpose Internet Mail Extensions (MIME):**
 - Solução desenvolvida para expandir o e-mail e permitir envio de conteúdo multimídia e caracteres especiais.
 - MIME mantém o formato da RFC 822, mas adiciona uma estrutura mais robusta ao corpo da mensagem e regras para mensagens especiais, permitindo compatibilidade com sistemas existentes.

2. Arquitetura do Sistema de Correio Eletrônico

- **Componentes Principais:**
 - **Agente do Usuário:** Interface que conecta o usuário ao sistema de e-mail, permitindo funções como envio e recebimento de mensagens, anexos e gerenciamento de caixa de entrada.
 - Exemplos de agentes: Mozilla Thunderbird, Microsoft Outlook, Eudora.

- **Agente de Transferência de Mensagens:** Responsável por garantir que as mensagens cheguem ao destino. Funciona como servidor intermediário.
 - Exemplos de servidores: Postfix, Zimbra, Exchange.
- **Processo de Envio e Recebimento:**
 - **Envio:** O usuário redige a mensagem no agente do usuário (ex: Outlook) e solicita seu envio.
 - **Transferência:** O agente de transferência do remetente recebe a mensagem, analisa e a encaminha ao agente de transferência do destinatário.
 - **Recepção:** A mensagem é armazenada na caixa de mensagens (mailbox) do destinatário até que ele a acesse através do agente de usuário.

3. Protocolos de Correio Eletrônico

- **SMTP (Simple Mail Transfer Protocol):**
 - Protocolo padrão para transmissão de e-mails na internet, definido pela **RFC 5321**.
 - Baseado no protocolo de transporte **TCP**, assegurando a entrega sem erros.
 - Funcionamento: O servidor SMTP aguarda conexões. Quando o cliente conecta-se, o servidor se apresenta e verifica a prontidão para receber mensagens.
 - **Interação SMTP:**
 - O cliente informa o remetente e o destinatário, e o servidor confirma se a transferência é válida.
 - Mensagem é enviada, recebida e o servidor confirma a entrega antes de encerrar a conexão.
- **POP3 (Post Office Protocol v3):**
 - Definido pela **RFC 1939**, permite o **download das mensagens** do servidor para leitura offline no dispositivo local.
 - O POP3 conecta-se ao servidor, realiza o download e pode deletar as mensagens após a transferência.
 - Utilizado em dispositivos onde o usuário acessa e-mails sempre a partir do mesmo local.
- **IMAP (Internet Message Access Protocol):**
 - Especificado pela **RFC 3501**, o IMAP permite **acesso direto às mensagens no servidor** sem a necessidade de transferi-las para o dispositivo local.
 - As mensagens podem ser lidas em qualquer dispositivo conectado, ideal para usuários que acessam o e-mail de diferentes locais.
 - Possibilita também a criação, exclusão e organização de caixas de correio no servidor.

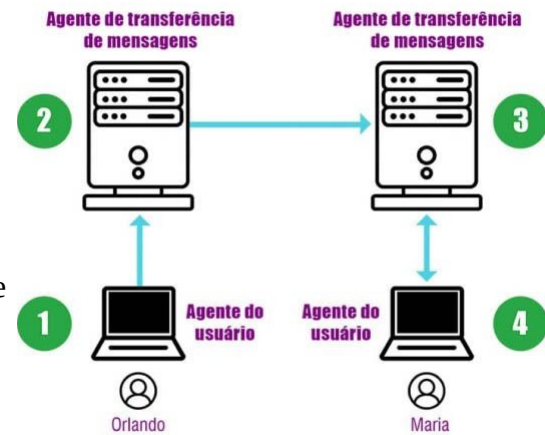
4. Webmail

- Não é um protocolo, mas uma interface que permite acesso ao e-mail diretamente de navegadores de internet.
- O usuário acessa suas mensagens após fazer login, similar ao funcionamento de um cliente **IMAP**.
- Facilita o acesso remoto sem a necessidade de configurar um cliente de e-mail específico no dispositivo.

Principais Conceitos e Funcionamento

- **SMTP:** Utilizado para transmissão e entrega no servidor de destino, com conexão inicial e confirmação de envio entre cliente e servidor.
- **POP3:** Para download e armazenamento local das mensagens, útil para acesso offline.
- **IMAP:** Para leitura das mensagens diretamente no servidor, ideal para múltiplos acessos em dispositivos diferentes.
- **Webmail:** Acesso ao e-mail via navegador, sem necessidade de protocolo específico.

Esse resumo detalhado cobre os aspectos fundamentais e mais técnicos dos sistemas de e-mail, facilitando a compreensão da arquitetura e dos protocolos envolvidos na comunicação eletrônica.



DNS

O **Domain Name System (DNS)** facilita a comunicação na internet convertendo nomes de domínio em endereços IP, organizando-os em uma estrutura hierárquica. Ele permite que usuários acessem hospedeiros por nomes ao invés de números binários, simplificando o processo e evitando a necessidade de atualizar constantemente endereços IP.

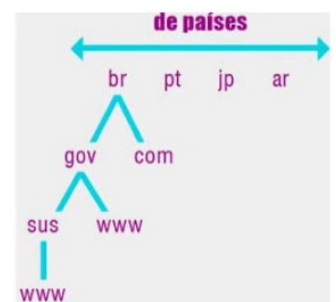
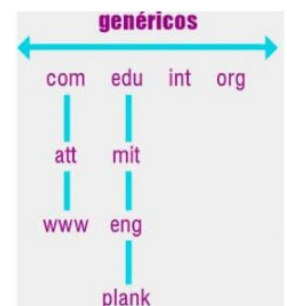
Principais serviços do DNS:

1. **Mapeamento de nomes para endereços IP** – Localiza o endereço IP correspondente ao nome de um hospedeiro.
2. **Identificação de servidores de e-mail** – Direciona emails para servidores específicos.
3. **Apelidos para hospedeiros** – Permite o uso de nomes alternativos para hospedeiros.
4. **Distribuição de carga** – Equilibra o tráfego entre múltiplos servidores.
5. **Mapeamento reverso** – Permite a identificação de um nome de hospedeiro a partir de um endereço IP.

Espaços de Nomes no DNS

O **espaço de nomes do DNS** é estruturado em níveis hierárquicos e dividido em domínios que podem ser **genéricos** ou **de países**:

- **Domínios Genéricos:** Associados ao tipo de organização:
 - **.com:** comercial
 - **.edu:** instituições educacionais
 - **.int:** organizações internacionais
 - **.org:** organizações sem fins lucrativos
- **Domínios de Países:** Cada país tem seu próprio domínio:
 - **.br:** Brasil
 - **.pt:** Portugal
 - **.jp:** Japão
 - **.ar:** Argentina



Regras e Estrutura dos Domínios:

- Nomes de domínio são formados pelo caminho até a raiz, separados por pontos.
- Subdomínios podem ser criados livremente com permissão do domínio principal.
- Não há distinção entre letras maiúsculas e minúsculas (ex.: EDU e edu são iguais).
- Limites: nomes de componentes até 63 caracteres, caminho completo até 255 caracteres.

Protocolo de Implementação:

- O DNS usa o **UDP (User Datagram Protocol)**, que é um protocolo de transporte não confiável, exigindo que o próprio software DNS gerencie a confiabilidade da comunicação.

Resolução de Nomes no DNS

O **espaço de nomes DNS** é organizado em **zonas**, que são independentes e têm pelo menos um **servidor de nomes principal** e um **servidor de nomes secundário**:

- **Servidor de Nomes Principal**: Armazena informações das zonas e repassa para servidores secundários.
- **Servidor de Nomes Secundário**: Atua como backup, respondendo pelas zonas se o servidor principal falhar.

Função das Zonas no DNS:

- As zonas definem o que um servidor deve resolver. Um **servidor autoritativo** é responsável pela resolução da zona sob sua jurisdição.

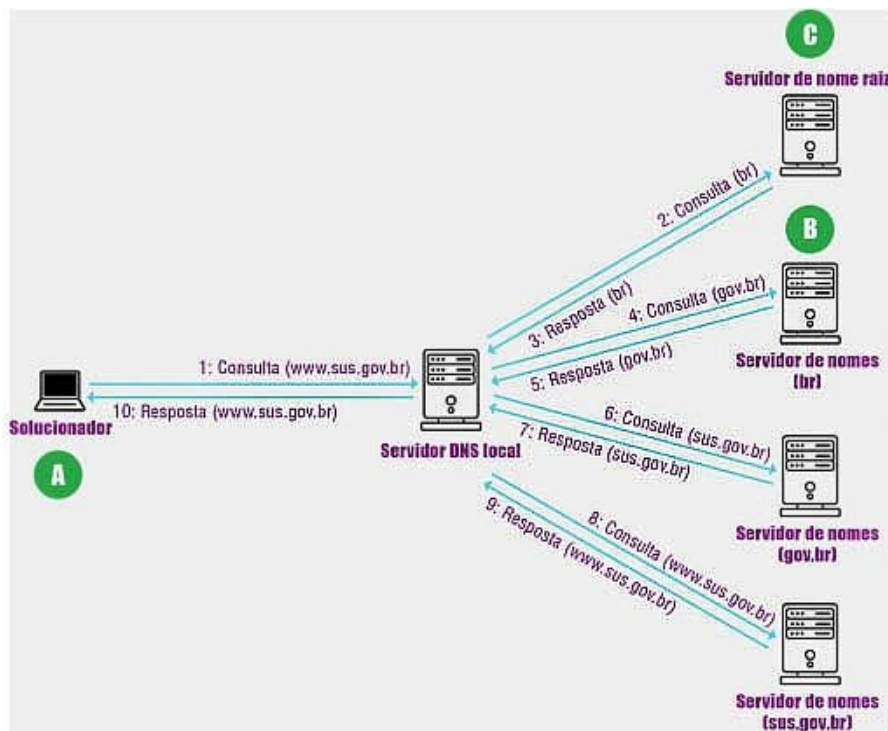
Componentes Principais do DNS:

1. **Registros de recursos**: Armazenados em um banco de dados distribuído.
2. **Servidores de Nomes DNS**: Mantêm zonas específicas.
3. **Solucionadores DNS (Clientes)**: Solicitam a resolução de nomes.

Processo de Resolução de Nomes:

- Quando um **solucionador** (cliente) solicita a resolução de um nome:
 - **Servidor é autoritativo** para a zona: Resolve e envia ao solucionador.
 - **Servidor não é autoritativo** mas possui o dado em cache: Envia a resolução em cache.
 - **Servidor não é autoritativo nem tem cache**: Realiza uma busca para resolver o nome.

Funcionamento do DNS



1. Consulta Recursiva:

- A aplicação do cliente solicita a resolução de um nome, como "www.sus.gov.br".
- O **solucionador envia a consulta ao servidor de nomes local**, que se responsabiliza por retornar a resposta completa, sem respostas parciais.

2. Consulta Iterativa:

- Se o servidor local não tem informações em cache, ele faz uma **consulta iterativa** para resolver o nome.
- Primeiro, consulta o **servidor raiz**, que indica o **servidor responsável pelo domínio de nível superior**, como ".br".
- O servidor local realiza consultas sequenciais até resolver o nome completo.

3. Prevenção de Sobrecarga:

- Para evitar excesso de consultas e sobrecarga, os servidores DNS devem restringir **consultas recursivas** apenas a clientes autorizados. Consultas de clientes não autorizados são automaticamente negadas.

Este processo de consultas recursivas e iterativas garante que o DNS funcione de forma eficaz, evitando sobrecarga nos servidores.

Camada de Transporte na Internet

A **camada de transporte** na arquitetura TCP/IP é fundamental para a comunicação eficaz entre computadores. Ela atua como intermediária, assegurando que as mensagens enviadas por aplicações (como e-mails ou streaming) cheguem ao destino correto, funcionando independente das redes físicas em uso.

Principais Funções da Camada de Transporte

- **Fornecer serviços para a camada de aplicação:** A camada de transporte facilita a comunicação para a camada acima, que é a de aplicação, oferecendo serviços específicos para tornar essa comunicação eficiente e confiável.
- **Serviços com ou sem conexão:** A camada de transporte oferece duas modalidades principais, permitindo que as aplicações escolham o serviço mais adequado às suas necessidades.

Tipos de Serviços

1. Serviço Orientado à Conexão (Confiável)

- **Objetivo:** Oferece um método seguro de transferência, garantindo que os dados cheguem corretamente ao destino.
- **Como Funciona:** Este serviço possui três fases (não detalhadas no texto) que gerenciam a conexão para verificar e corrigir erros. Pacotes são retransmitidos se houver falhas, assegurando a chegada exata dos dados.
- **Aplicações Relevantes:** É usado em casos onde a integridade dos dados é crucial, como em transferências de arquivos e e-mails, pois estes demandam que o conteúdo chegue sem erros.

2. Serviço Sem Conexão (Não Confiável)

- **Objetivo:** Entregar pacotes de forma rápida, sem garantia de chegada ou verificação de erro.
- **Como Funciona:** Este serviço não realiza controle sobre os pacotes enviados; assim, se um pacote se perde ou contém erro, ele não será retransmitido.
- **Vantagem:** A ausência de verificação reduz o overhead, tornando o serviço mais rápido.
- **Aplicações Relevantes:** É ideal para situações em que a velocidade é mais importante que a precisão, como em chamadas de voz, onde pequenos ruídos são toleráveis, mas atrasos prejudicam a experiência.

Considerações sobre o Uso de Cada Serviço

As aplicações escolhem entre os dois tipos de serviço baseando-se em suas necessidades específicas:

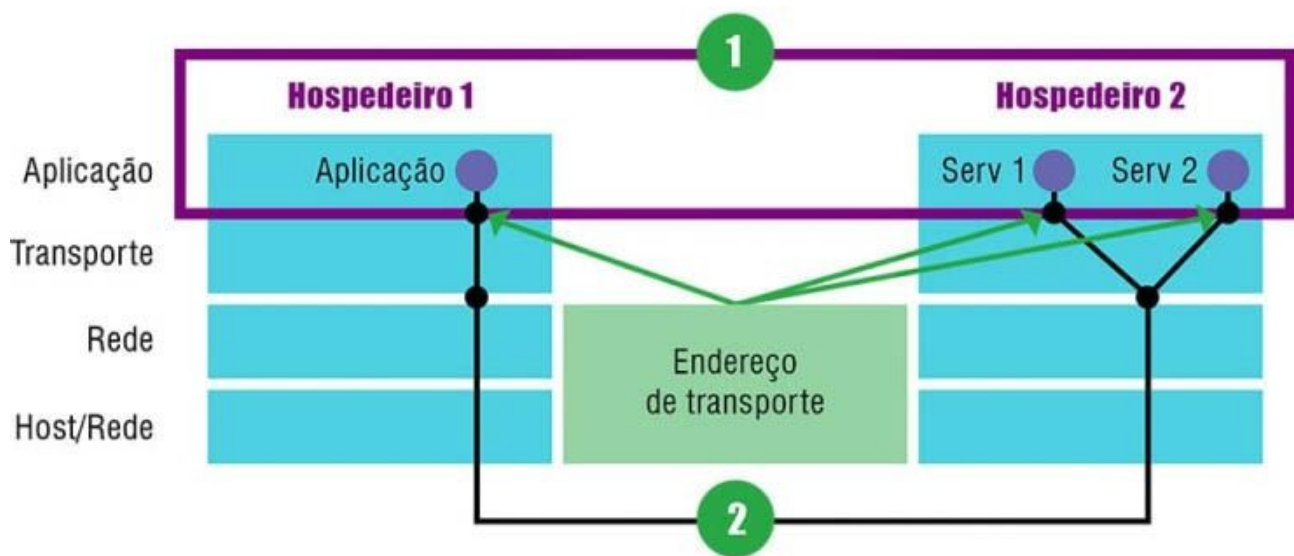
- **Confiabilidade vs. Desempenho:** Aplicações que exigem dados precisos, como e-mails, preferem o serviço orientado à conexão. Já aplicações que priorizam a velocidade, como telefonia, preferem o serviço sem conexão, aceitando possíveis ruídos ou pequenas perdas de dados.

A camada de transporte, portanto, desempenha um papel crucial na comunicação digital, ajustando-se às demandas de confiabilidade e eficiência para cada tipo de aplicação.



Endereçamento dos Processos (Camada de Transporte)

- **Objetivo do Endereçamento:** Permite que uma mensagem enviada pelo sistema seja entregue à aplicação específica no hospedeiro remoto. Isso é essencial para distinguir entre várias aplicações que podem estar rodando no mesmo servidor.
- **Identificação de Aplicações:** Cada mensagem do protocolo de transporte inclui o "endereço de transporte" da aplicação, o que permite ao hospedeiro destino identificar para qual aplicação os dados devem ser entregues.
- **Protocolos e Endereços de Transporte:** No TCP/IP, o **TCP** e o **UDP** utilizam portas para identificar o endereço de transporte de cada aplicação, facilitando a comunicação entre aplicações específicas nos hospedeiros.
- **Métodos de Associação de Endereços:**
 1. **Endereços Estáveis:** Certos serviços possuem endereços fixos conhecidos, que são distribuídos aos usuários quando se associam à rede, garantindo fácil identificação.
 2. **Servidor de Nomes:** Um processo alternativo que associa o nome do serviço a um endereço de transporte, usado para localizar o endereço correspondente a determinado serviço quando necessário.
- **Cabeçalhos da Camada de Transporte:** Contêm informações essenciais, como o endereço de transporte (porta), que orientam a entrega das mensagens à aplicação correta.



Multiplexação e Demultiplexação

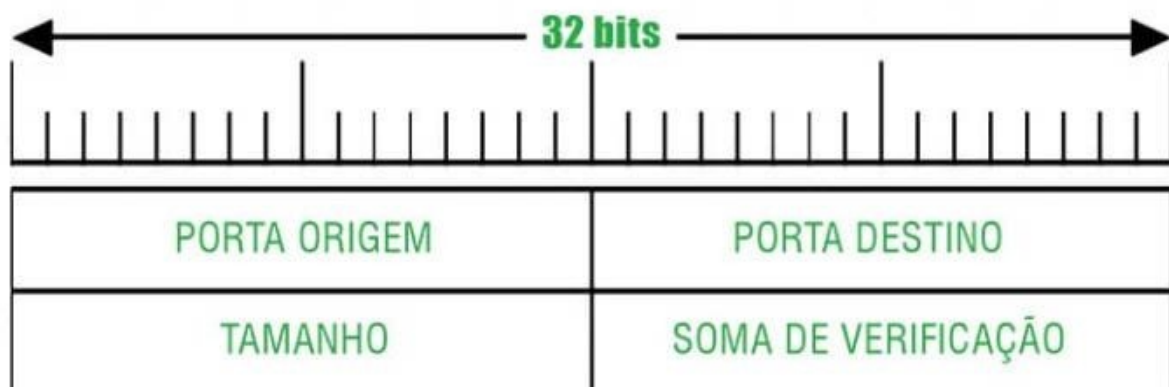
- **Função:** Multiplexação e demultiplexação fornecem entrega de dados entre processos de aplicações em diferentes hospedeiros. A camada de transporte usa essas técnicas para gerenciar o fluxo de dados entre várias aplicações simultaneamente.
- **Processo de Demultiplexação:** No hospedeiro destino, a camada de transporte recebe segmentos da camada de rede e entrega cada um à aplicação correta usando o número da porta. O cabeçalho do segmento contém campos de endereçamento que identificam a porta de destino, facilitando a entrega.
- **Processo de Multiplexação:** No hospedeiro de origem, dados de diferentes portas são reunidos e encapsulados com informações de cabeçalho, criando segmentos para envio à camada de rede.
- **Exemplo Prático (Caso de Eduardo):**
 - Eduardo utiliza vários programas (navegador, e-mail e transferência de arquivos) que compartilham o protocolo TCP.
 - Cada programa se registra em uma porta específica (ex.: navegador na porta 11278), permitindo ao TCP distinguir entre eles e encaminhar as respostas corretamente.
- **Multiplexação e Demultiplexação em Ação:**
 - **Multiplexação:** Associa mensagens enviadas pelas aplicações às suas respectivas portas, permitindo o uso simultâneo da rede por várias aplicações.
 - **Demultiplexação:** Recebe mensagens e direciona-as ao processo apropriado pela identificação da porta destino no segmento, assegurando a entrega para a aplicação correta.

Protocolos de Transporte da Internet

- **Contexto:** Os protocolos de transporte da Internet, como TCP e UDP, oferecem serviços essenciais para a comunicação entre redes de computadores.
- **UDP (User Datagram Protocol):**
 - Protocolo simples e eficiente.
 - Focado em agilidade e entrega rápida.
 - Ideal para aplicações que precisam de rapidez sem controle de erros detalhado.
- **TCP (Transmission Control Protocol):**
 - Protocolo completo e confiável.
 - Garante entrega sem erros, independentemente da qualidade da rede.
 - Recomendado para aplicações que necessitam de alta confiabilidade na transmissão.
- **Portas:** Mecanismos que permitem a identificação e comunicação entre diferentes aplicações na Internet, facilitando a conexão entre elas através de protocolos como TCP e UDP.

Protocolo UDP

- **Definição e Funcionamento:**
 - UDP (User Datagram Protocol) é um protocolo de transporte simples, rápido e sem conexão.
 - Atua na entrega direta entre a camada de aplicação e a de rede, sem garantir confiabilidade na transmissão.
- **Características Principais:**
 - **Multiplexação e Demultiplexação:** Utiliza campos de portas de origem e destino para entregar as mensagens ao processo correto no destino.
 - **Verificação de Erros:** Implementa um campo de soma de verificação (CRC) para checar erros na mensagem; se houver erro, o segmento pode ser descartado ou entregue com aviso.
 - **Tamanho do Segmento:** O segmento UDP pode ter até 64 KB (65.536 bytes), definido pelo campo de tamanho.
- **Protocolo Sem Estado:**
 - UDP é sem estado e não orientado à conexão, adequado para transmissões rápidas ou pequenas que não precisam de confiabilidade.
 - A responsabilidade pela garantia de entrega recai sobre o programador da aplicação.
- **Exemplos de Aplicações que Usam UDP:**
 - DNS (Domain Name System)
 - SNMP (Simple Network Management Protocol)
 - TFTP (Trivial File Transfer Protocol)
 - RPC (Remote Procedure Call)



Cabeçalho TCP

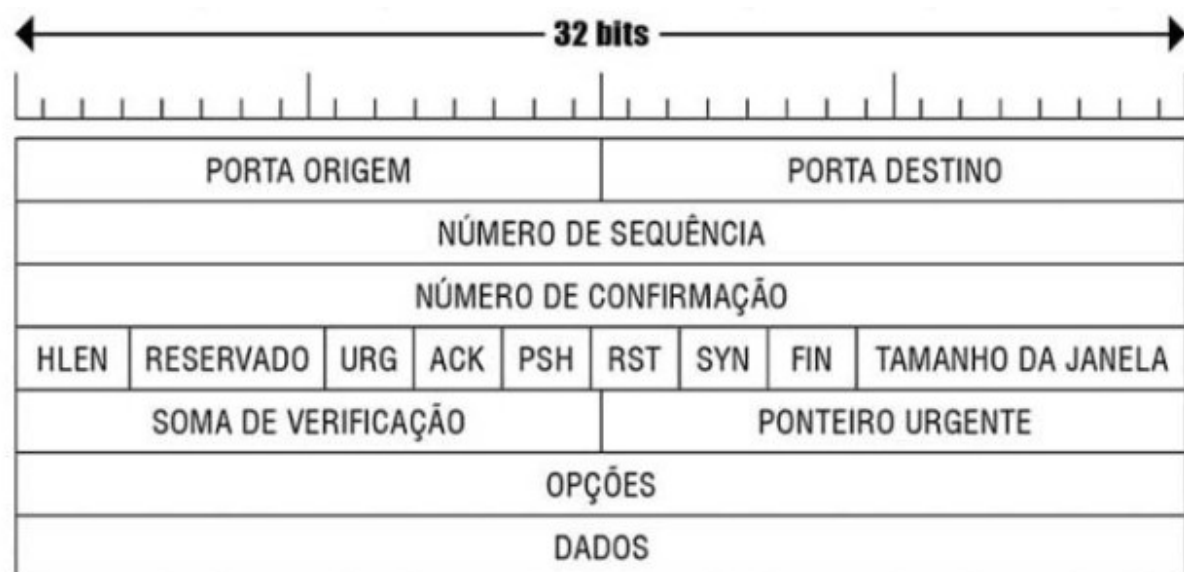


Tabela: 32 Bits.

O cabeçalho TCP possui 20 bytes fixos, podendo ser seguido por até 65.515 bytes de dados, se aplicável. Segmentos sem dados geralmente são usados para confirmações e controle. A seguir, são descritos os campos do cabeçalho TCP:

1. **Porta origem e porta destino:** Identificam os processos de origem e destino nos hospedeiros, permitindo a multiplexação e demultiplexação das aplicações.

2. **Número de sequência:** Indica a posição do byte dos dados, indicando onde ele deve ser colocado na sequência.
3. **Número de confirmação:** Especifica o próximo byte esperado no fluxo contrário. Implica que todos os bytes anteriores foram recebidos corretamente.
4. **HLEN (Header Length):** Tamanho do cabeçalho em palavras de 32 bits.
5. **Reservado:** Campo não utilizado, valores preenchidos são desconsiderados.
6. **Campo de flags de 1 bit cada:**
 - **URG:** Aponta que o segmento contém dados urgentes.
 - **ACK:** Indica que o campo de número de confirmação é válido.
 - **PSH:** Solicita a entrega imediata dos dados à aplicação, sem armazená-los em buffer.
 - **RST:** Usado para reinicializar uma conexão ou rejeitar um segmento inválido.
 - **SYN:** Usado para estabelecer conexões.
 - **FIN:** Indica o encerramento de uma conexão.
7. **Tamanho da janela:** Define quantos bytes podem ser enviados após o byte confirmado, controlando o fluxo de dados.
8. **Soma de verificação:** Verifica a validade dos dados e cabeçalho, detectando erros.
9. **Ponteiro urgente:** Indica a parte dos dados que é urgente, válido apenas se o flag URG estiver ativo.
10. **Opções:** Oferecem recursos extras não previstos no cabeçalho comum.
11. **Dados:** Contém os dados enviados pela camada superior, que serão entregues ao destino.

Esses campos são essenciais para garantir a comunicação confiável e a integridade dos dados no protocolo TCP.

Conexão TCP

A conexão TCP é estabelecida através do processo conhecido como **three-way handshake**, um protocolo de troca de mensagens entre o cliente e o servidor para garantir a criação de uma conexão confiável. O processo ocorre da seguinte maneira:

1. Início da Conexão (Cliente -> Servidor):

- O **cliente** envia um segmento TCP com o **bit SYN** ativado. O **SYN** (synchronize) é usado para iniciar a solicitação de uma conexão, e esse segmento informa ao servidor que o cliente deseja estabelecer uma comunicação. O segmento também tem o **bit ACK** (acknowledgment) desativado, indicando que o cliente ainda não recebeu uma confirmação do servidor.

2. Resposta do Servidor (Servidor -> Cliente):

- O **servidor** verifica se há um processo (aplicação) aguardando conexões na porta especificada pelo cliente. Se houver, o servidor responde com um segmento TCP que tem tanto o **bit SYN** quanto o **bit ACK** ativados. O **ACK** é utilizado para confirmar a recepção do pedido de conexão e indicar que o servidor está pronto para estabelecer a comunicação.
- Se não houver um processo aguardando na porta, o servidor envia um segmento com o **bit RST** (reset) ativado, rejeitando a conexão e informando ao cliente que a porta está indisponível.

3. Confirmação do Cliente (Cliente -> Servidor):

- O **cliente** envia um segmento TCP de volta com o **bit ACK** ativado para confirmar a recepção da resposta do servidor. Isso completa o processo de estabelecimento da conexão.

Processo de Transmissão e Controle:

- Durante o **three-way handshake**, cada segmento TCP contém **números de sequência** que ajudam a identificar a ordem dos pacotes e garantir que os dados sejam recebidos corretamente. O TCP é um protocolo **orientado à conexão**, ou seja, ele assegura que todos os pacotes enviados sejam confirmados, reorganizados e retransmitidos em caso de erro, garantindo uma comunicação confiável.
- O **controle de fluxo** no TCP é feito por meio de um mecanismo chamado **janela deslizante**. A janela controla quantos dados podem ser enviados antes que o transmissor receba uma confirmação. Isso previne a sobrecarga de dados e assegura uma comunicação eficiente.
- O **bit SYN** é fundamental para o estabelecimento da conexão, o **bit ACK** confirma a recepção e o **bit RST** é usado para reiniciar ou rejeitar conexões. O uso adequado desses bits no cabeçalho TCP garante que o fluxo de dados entre o cliente e o servidor seja controlado e seguro.

Política de Transmissão TCP:

- O TCP também lida com a **perda de segmentos** e o **controle de fluxo**. Caso um segmento se perca ou seja corrompido, o TCP retransmite o pacote. O controle de fluxo evita que o transmissor envie dados mais rapidamente do que o receptor pode processar.

Em resumo, o **three-way handshake** é o mecanismo fundamental para a criação de uma conexão TCP confiável entre dois processos, garantindo que ambos os lados estejam preparados para a comunicação. Esse processo é essencial para assegurar que a transmissão de dados ocorra de maneira ordenada e sem erros.

Conexão TCP

A conexão TCP é estabelecida através do processo de **three-way handshake**. Durante esse processo:

1. Estabelecimento de Conexão:

- O **cliente** solicita a conexão, enviando um segmento TCP com o **bit SYN** ativado e o **bit ACK** desativado, especificando o **endereço IP** e a **porta** do destino.
- O **destino** verifica se há um processo aguardando na porta especificada:
 - Se não houver processo, o destino envia um segmento com o **bit RST** ativado, rejeitando a conexão.
 - Se houver um processo, o segmento é entregue a ele, que pode aceitar ou rejeitar a conexão. Se aceitar, o **bit ACK** é ativado no retorno.

2. Política de Transmissão TCP: A transmissão TCP envolve o controle de fluxo e o gerenciamento de perdas de segmentos, com a política tratando da recuperação em caso de falhas.

Tipos de Portas

Portas Conhecidas:

- **Definição e Função:** Portas são identificadores numéricos que permitem ao protocolo de transporte direcionar pacotes de dados para a aplicação correta no destino, assegurando que a mensagem chegue à aplicação específica.
- **Como identificar a porta de uma aplicação?**
 1. **Registro dinâmico:** A aplicação é registrada sempre que inicia, permitindo que o cliente consulte sua porta.
 2. **Porta fixa:** Algumas aplicações utilizam sempre a mesma porta para facilitar a comunicação, com a troca de mensagens ocorrendo em um endereço pré-definido.
- **Número de Porta:** Tanto o TCP quanto o UDP identificam as portas com números inteiros de 16 bits.
- **Porta Conhecida:** Algumas aplicações utilizam portas fixas e conhecidas, chamadas "portas conhecidas", que são definidas por padrões e podem ser consultadas online através do repositório da RFC 3232.
- **Tabela de Portas Conhecidas:**
 1. **Porta 7:** echo
 2. **Porta 20:** ftp-data
 3. **Porta 21:** ftp
 4. **Porta 22:** ssh
 5. **Porta 23:** telnet
 6. **Porta 25:** smtp
 7. **Porta 53:** domain
 8. **Porta 69:** tftp
 9. **Porta 80:** http
 10. **Porta 110:** pop-3
 11. **Porta 119:** nntp
 12. **Porta 161:** snmp
 13. **Porta 162:** snmp-trap
 14. **Porta 443:** https

Evolução das Aplicações e Comunicação via Sockets

Evolução das Aplicações:

- **Primeiras aplicações:** Inicialmente, as aplicações eram executadas localmente, em um único computador, sem a necessidade de comunicação com outros sistemas ou recursos externos.
- **Mudança com as redes de computadores:** À medida que as redes se popularizaram, as aplicações começaram a trocar informações através da rede, deixando de ser executadas isoladamente no computador local.

- **Crescimento da Internet:** Com a explosão da Internet, a troca de dados entre as aplicações tornou-se fundamental. Hoje, as informações de uma aplicação não estão restritas ao dispositivo local, mas podem estar distribuídas por diversos dispositivos conectados na rede.

Compreensão de Comunicação de Dados:

- **Objetivo:** A troca de informações entre programas de aplicação é essencial e será abordada através de **sockets** em Python, que possibilitam a comunicação entre aplicações via rede.
- **Tópicos a serem explorados:**
 1. Definição de **socket** e tipos de sockets.
 2. Implementações de sockets com os protocolos **TCP** e **UDP**.
 3. Considerações para o desenvolvimento de **aplicações seguras**.

Protocolos e a Camada de Aplicação

Arquitetura em Camadas:

- **Importância dos serviços de rede:** Dispositivos como computadores, notebooks e celulares precisam de serviços para serem úteis, e esses serviços são fornecidos pelas aplicações de rede, que trocam pacotes de dados para comunicar informações.
- **Camada de Aplicação:** As aplicações de rede operam na camada de aplicação da arquitetura TCP/IP (5 camadas) ou modelo OSI (7 camadas). No modelo TCP/IP, as camadas são:
 1. Físico
 2. Enlace
 3. Rede
 4. Transporte
 5. Aplicação

Já o modelo OSI é mais detalhado, incluindo camadas como apresentação e sessão.

Elementos da Arquitetura em Camadas:

- **Serviços:** Cada camada requisita serviços da camada inferior e oferece serviços para a camada superior, criando a comunicação vertical entre camadas adjacentes.
- **Protocolos:** São regras que guiam a comunicação entre camadas de máquinas diferentes, garantindo a troca de dados eficiente e conforme as convenções estabelecidas.
- **Interfaces:** Ponto de comunicação entre camadas adjacentes, onde cada camada define os serviços e informações para a camada superior.

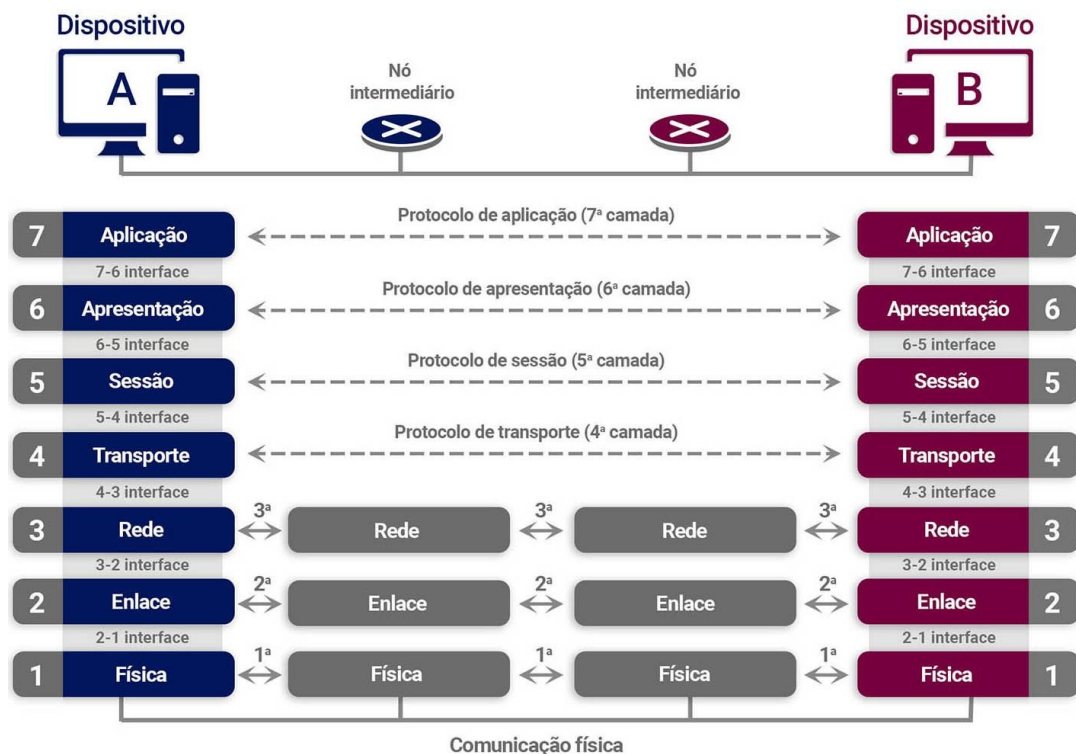
Serviços da Camada de Aplicação:

- **Função da camada de aplicação:** Oferece aos usuários acesso a serviços como transferência de arquivos, e-mails, chats, jogos, etc. Os desenvolvedores atuam nesta camada, criando a interface de aplicação e determinando como a troca de informações será realizada.

- **Desenvolvimento de protocolos:** Os desenvolvedores de aplicações de rede são responsáveis por criar protocolos para a camada de aplicação. Embora muitos desenvolvedores usem protocolos existentes, como HTTP para aplicações web, entender o funcionamento desses protocolos é fundamental para um bom profissional de TI.

Reflexão Final:

- **Entendimento do desenvolvimento de protocolos:** Para quem já desenvolve aplicações, é importante compreender como a comunicação entre aplicações ocorre de forma interna, além de apenas utilizar protocolos como HTTP, para garantir uma atuação mais eficaz no campo das TI.



Interface Socket e Comunicação das Aplicações

Socket e a Interface API de Comunicação:

- **Função do Socket:** O socket atua como uma interface entre a camada de aplicação e a camada de transporte, permitindo que aplicações troquem dados entre si em diferentes hosts. Essa interface é implementada pelo sistema operacional como uma API (Interface de Programação de Aplicação).
- **Arquitetura em Camadas:** Para a comunicação funcionar, cada camada de rede executa serviços para a camada acima dela. A camada de aplicação, por exemplo, consome os serviços da camada de transporte para enviar mensagens.
- **Funcionamento do Socket:** No modelo de comunicação via socket, o processo de envio transmite a mensagem "através de uma porta" (socket) até o processo receptor, que usa seu socket para receber a mensagem e reagir de acordo. Esse processo facilita a transferência de dados de um host de origem para um host de destino.

- **Controle dos Componentes:** O desenvolvedor controla os processos da camada de aplicação, enquanto o sistema operacional gerencia os protocolos de transporte TCP/IP, garantindo que os dados cheguem ao destino com segurança e integridade.



Serviços da Camada de Transporte

A camada de transporte oferece dois serviços fundamentais para a comunicação entre hosts:

1. **Multiplexação:** Identifica pacotes de diferentes serviços (como Web e e-mail) e os separa para que cada processo receba apenas os pacotes destinados a ele.
2. **Transporte Confiável:** Garante a entrega integral dos dados, gerenciando a retransmissão de pacotes perdidos, reordenando pacotes fora de sequência e eliminando duplicações.

Principais protocolos de transporte na Internet:

- **UDP (User Datagram Protocol):** Proporciona números de porta para identificação de serviços, mas não lida com perda, duplicação ou ordenação de pacotes.
- **TCP (Transmission Control Protocol):** Proporciona números de porta e assegura transmissão confiável, garantindo que os dados cheguem ao destino na sequência correta e sem perdas.

Esses protocolos permitem uma comunicação eficiente e ordenada entre aplicações de rede.

Multiplexação dos Fluxos de Rede

A multiplexação permite que várias comunicações compartilhem o mesmo meio de rede. Isso é feito através das portas: a porta de origem identifica o processo que enviou o pacote, e a porta de destino indica o aplicativo que receberá a comunicação no IP de destino.

A IANA organiza as portas em três categorias:

1. **Portas Bem Conhecidas (0–1023):** Reservadas para serviços essenciais; geralmente, os sistemas operacionais restringem seu uso a programas específicos.
2. **Portas Registradas (1024–49151):** Qualquer usuário pode usá-las, mas a IANA recomenda manter o uso para os serviços designados.
3. **Portas Dinâmicas (49152–65535):** Usadas livremente para atribuições arbitrárias em conexões temporárias.

Endereçamento do Socket

Para possibilitar a troca de mensagens, as aplicações devem ser endereçadas de forma exclusiva. Isso é feito através do **endereçamento de socket**, que combina o endereço IP e o número de porta, formando um identificador único para o processo cliente e o processo servidor.

A camada de transporte usa pares de endereços socket para enviar dados, com os endereços IP no cabeçalho IP e os números de porta nos cabeçalhos UDP ou TCP:

- **UDP (não orientado à conexão):** O processo emissor inclui o socket de destino (IP e porta) e o socket de origem para possibilitar a resposta.
- **TCP (orientado à conexão):** Estabelece uma conexão antes da troca de dados, conectando os sockets cliente e servidor. Após a conexão, a transmissão de dados ocorre pela própria conexão associada aos sockets.

A IANA mantém uma lista atualizada de portas lógicas em seu site para consulta.

Primitivas de Sockets (Berkeley Socket)

Para a comunicação entre processos em uma arquitetura cliente-servidor, são usadas primitivas de sockets no UNIX Berkeley para o protocolo TCP. Estas primitivas permitem criar, configurar e gerenciar conexões:

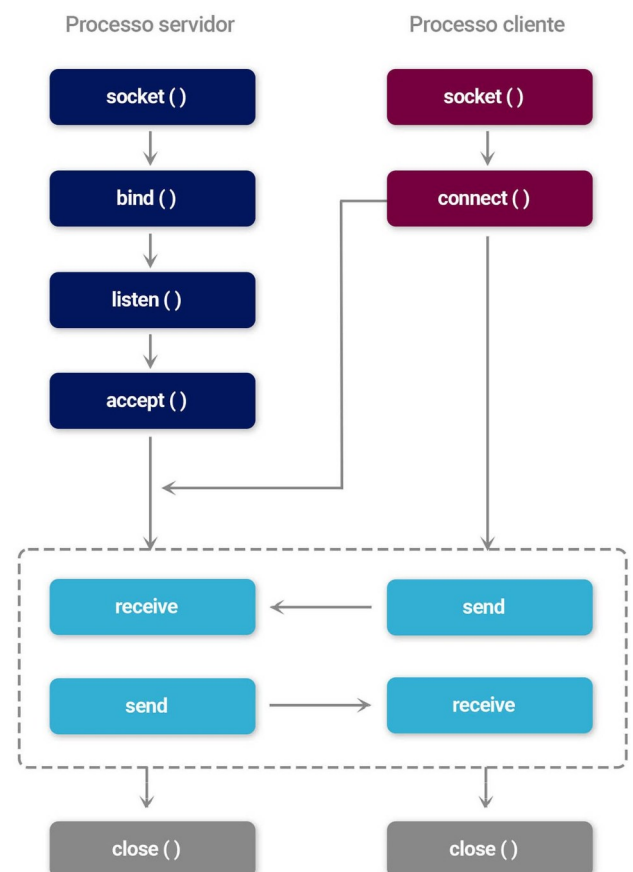
1. Servidor:

- **SOCKET:** Cria um socket.
- **BIND:** Liga o socket a uma porta.
- **LISTEN:** Disponibiliza o socket para aceitar conexões.
- **ACCEPT:** Espera conexões de clientes.
- Após estabelecer a conexão, usa **RECEIVE** para receber e **SEND** para responder. No final, a conexão é encerrada com **CLOSE**.

2. Cliente:

- **SOCKET e CONNECT:** Inicia o socket e tenta se conectar ao servidor.
- Após a conexão, usa **SEND** para enviar e **RECEIVE** para receber dados, finalizando com **CLOSE**.

Essas primitivas possibilitam o fluxo de dados bidirecional entre cliente e servidor para a execução dos serviços.



Desenvolvendo Sockets UDP em Python

- **Python 3** é uma linguagem de programação interpretada, orientada a objetos, de alto nível e com semântica dinâmica. Sua tipagem dinâmica e estruturas de dados de alto nível a tornam ideal para o desenvolvimento rápido de aplicativos e scripts.
- **Biblioteca socket:** O Python oferece a biblioteca `socket`, que fornece uma interface baseada em objetos para realizar chamadas de rede de baixo nível em sistemas compatíveis com POSIX, facilitando tarefas de rede.

- **Objetivo:** O conteúdo apresenta um exemplo de um servidor e um cliente UDP simples, utilizando a biblioteca `socket`.

Execução do código:

- Para executar como **servidor**:
`$ python udp_local.py servidor`
- Para executar como **cliente**:
`$ python udp_local.py cliente`
- A função `main` do programa determina qual função (servidor ou cliente) será executada com base na escolha feita ao rodar o comando.

Código do Socket para o Lado Servidor

- **Criação do Socket:**
O servidor começa criando um socket UDP com `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`. O socket é configurado para a família de protocolos **AF_INET** (Internet) e tipo **SOCK_DGRAM** (UDP). Nesse ponto, o socket ainda não está vinculado a nenhum endereço IP ou porta.
- **Vinculação do Socket:**
O comando `sock.bind(('127.0.0.1', porta))` associa o socket ao endereço IP e à porta UDP. Aqui, o IP é **127.0.0.1** (loopback), o que significa que o servidor só aceitará conexões da mesma máquina. Se a porta já estiver em uso, o programa gera um erro: `OSError: [Errno 98] Endereço já em uso`.
- **Exibição do Status:**
O comando `print('Servidor >> Escutando no IP e porta {}'.format(sock.getsockname()))` exibe uma mensagem no terminal para informar que o servidor está em execução.
- **Recepção de Dados:**
O servidor entra em um loop infinito com `while True:`, aguardando dados com `sock.recvfrom(MAX_BYTES)`, onde `MAX_BYTES` define o tamanho máximo do datagrama UDP (65.535 bytes). O comando `recvfrom()` retorna o endereço do cliente e o conteúdo em formato de bytes.
- **Processamento da Mensagem:**
O conteúdo recebido é decodificado em string com `data.decode('ascii')`. Em seguida, o servidor exibe a mensagem no console com `print('Servidor >> O cliente no IP e porta {} enviou a mensagem {!r}'.format(address, text))`.
- **Envio de Resposta:**
O servidor envia uma resposta ao cliente com o comando `sock.sendto(data, address)`, onde `address` é o endereço do cliente e `data` é o conteúdo da resposta.
- **Exemplo de Uso:**
O código recebe uma mensagem do cliente e responde com o comprimento da mensagem.

Em programas mais complexos, entre `recvfrom()` e `sendto()` haveria a lógica do programa, como o processamento de requisições HTTP (verificando métodos como GET, POST, etc.).

Código do Socket UDP para o Lado Cliente

- **Execução do Servidor:**

O servidor é iniciado com o comando:

```
ruby
Copiar código
$ python udp_local.py servidor
```

O servidor exibe uma mensagem indicando que está escutando no IP e porta:

```
java
Copiar código
Servidor >> Escutando no IP e porta ('127.0.0.1', 1060)
```

Ele ficará aguardando mensagens do cliente.

- **Criação do Socket no Cliente:**

O cliente cria um socket UDP com:

```
python
Copiar código
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

O socket é utilizado para enviar e receber mensagens.

- **Preparação da Mensagem:**

A mensagem a ser enviada é preparada com:

```
python
Copiar código
text = 'Mensagem para o servidor: Hora atual {}'.format(datetime.now())
```

Em seguida, é codificada em formato ASCII:

```
python
Copiar código
data = text.encode('ascii')
```

- **Envio da Mensagem para o Servidor:**

A mensagem é enviada para o servidor com:

```
python
Copiar código
sock.sendto(data, ('127.0.0.1', porta))
```

O comando `sendto()` envia o datagrama para o servidor, com o endereço e a porta especificados.

- **Recepção de Resposta do Servidor:**

O cliente aguarda a resposta do servidor com:

```
python
Copiar código
data, address = sock.recvfrom(MAX_BYTES)
```

O conteúdo da resposta é armazenado em `data` e o endereço do servidor em `address`. O cliente então exibe a resposta no terminal.

- **Execução do Cliente:**

Quando o cliente é executado, ele exibe informações sobre o endereço e a porta usados:

```
java
Copiar código
Cliente >> O sistema operacional do cliente informou o IP e porta
('0.0.0.0', 46056)
```

E exibe a resposta do servidor:

```
java
Copiar código
Cliente >> O servidor ('127.0.0.1', 1060) respondeu 'Mensagem para o
cliente: O dado enviado possui comprimento de 63 bytes'
```

- **Relato no Servidor:**

O servidor exibe mensagens informando os detalhes das conexões dos clientes:

```
java
Copiar código
Servidor >> O cliente no IP e porta ('127.0.0.1', 46056) enviou a mensagem
'Mensagem para o servidor: Hora atual 2022-11-15 23:08:56.025842'
```

- **Lógica do Programa:**

Antes de `sendto()`, a lógica no cliente prepara a mensagem a ser enviada. Após `recvfrom()`, o cliente trata a mensagem recebida. Em programas mais complexos, como clientes HTTP, a lógica pode incluir o tratamento de códigos de resposta (ex: 200 OK, 404 Not Found).

- **Exercício Proposto:**

O código pode ser modificado para implementar outras lógicas, como jogos remotos (ex: jogo da velha).

Sockets TCP em Python

- **Protocolo UDP:**

O **UDP** é um protocolo não orientado à conexão e sem confiabilidade, sendo inadequado para aplicações que exigem garantia na entrega de dados, como transferência de arquivos e acesso a páginas Web.

- **Protocolo TCP:**

Para aplicações que requerem confiabilidade, o **TCP** é recomendado. Ele utiliza um **socket do tipo stream** e garante a entrega confiável dos dados, mantendo a ordem e sem duplicações. O desenvolvedor não precisa se preocupar com esses detalhes, pois o TCP cuida da entrega dos dados.

- **Uso do UDP com Confiabilidade:**

Caso se deseje usar o **UDP** por ser mais rápido e simples, a confiabilidade deve ser

implementada **na camada de aplicação**, ou seja, o desenvolvedor terá que garantir a entrega dos dados por conta própria.

- **Características do Socket TCP:**

1. **Orientado à Conexão:** A troca de dados entre cliente e servidor exige o estabelecimento de uma conexão.
2. **Fluxo de Bytes:** O TCP trata os dados como um fluxo de bytes, podendo enviar mensagens completas ou apenas partes delas.

- **Estabelecimento de Conexão:**

No TCP, o processo de troca de dados não é direto. Ele envolve uma fase de **estabelecimento de conexão**, onde o cliente envia um pedido e o servidor aceita ou rejeita esse pedido. O processo de abertura de conexão é feito através do **Three Way Handshake**, que não será detalhado neste estudo.

Diferenças entre os Sockets TCP e UDP

- **Execução e Etapas do TCP e UDP:**

O código para **TCP** e **UDP** segue o mesmo formato de cliente e servidor, com a escolha do tipo de execução sendo feita na linha de comando. As principais diferenças entre os dois protocolos são destacadas no código exemplo "tcp_exemplo.py".

- **Estabelecimento de Conexão no TCP:**

No **TCP**, o método `sock.connect((host, port))` inicia o **Three-Way Handshake** para garantir a conexão entre o cliente e o servidor. Se a conexão não for possível (exemplo: servidor não ativo), uma mensagem de erro é gerada (e.g., `ConnectionRefusedError`).

- **Simplicidade no Cliente TCP:**

O cliente **TCP** é mais simples, pois o protocolo cuida da retransmissão de pacotes em caso de falhas. O cliente pode enviar dados sem se preocupar com a perda de pacotes, enquanto no **UDP**, o cliente precisa lidar com pacotes perdidos e garantir a confiabilidade.

- **Diferenças de Envio de Dados:**

- **UDP** usa os comandos `sendto()` e `recvfrom()`, enviando **datagramas** independentes. Cada datagrama é enviado ou recebido como uma unidade completa, sem divisão.
- **TCP**, por outro lado, transmite dados como um fluxo contínuo de bytes, podendo dividir os dados em pacotes de tamanhos variáveis. O código precisa estar preparado para reagrupá-los no receptor.

- **Comportamento de `send()` e `recv()` no TCP:**

- O método `send()` pode bloquear o programa, se o buffer de saída estiver cheio, ou realizar a transmissão parcial dos dados, retornando o número de bytes enviados. Para garantir o envio completo, o código usa o método `sendall()`, que cuida de transmitir todos os dados.

- O método **recv()** funciona de forma semelhante, podendo bloquear até que os dados estejam disponíveis. Para garantir que uma mensagem completa seja recebida, **recv()** deve ser chamado dentro de um **loop**.
- **Função `recvall()`:**
A função **recvall()** é definida para garantir que todos os dados sejam recebidos corretamente, com um loop que verifica se a mensagem completa foi recebida.
- **Considerações sobre `sendall()` e `recv()`:**
A biblioteca padrão do Python não tem um equivalente direto de **sendall()** para o método **recv()**, pois, em muitos casos, as mensagens não são de tamanho fixo, tornando necessário um controle mais complexo para processar os dados recebidos.

Pontos-chave:

- **TCP** oferece uma conexão confiável, com controle de fluxo e retransmissão de pacotes.
- **UDP** é mais rápido e simples, mas não garante entrega ou ordem dos pacotes.
- **TCP** requer o Three-Way Handshake e um controle mais cuidadoso no envio e recebimento dos dados.
- **UDP** envia datagramas independentes e o código precisa garantir a confiabilidade.

Detalhamento da aplicação prática de sockets com TCP

1. Tipos de Sockets de Fluxo:

- **Socket de escuta:** Usado por servidores para disponibilizar uma porta para conexões de entrada.
- **Socket conectado:** Representa a comunicação ativa entre um servidor e um cliente específico.

2. Etapas de um Programa com TCP:

- O servidor executa `bind()` para vincular uma porta e, em seguida, `listen()` para aguardar conexões.
- Após `listen()`, o servidor utiliza `accept()` para aceitar a conexão de um cliente, retornando um novo socket para gerenciar a comunicação com esse cliente.
- `getsockname()` retorna o endereço local do socket e `getpeername()` retorna o endereço do cliente conectado.

3. Fluxo do Código (Exemplo `tcp_exemplo.py`):

- **Servidor:** Após execução de `bind()` e `listen()`, aguarda a conexão de um cliente. Exemplo:

```
java
Copiar código
$ tcp_exemplo.py servidor "127.0.0.1"
Escutando em ('127.0.0.1', 1060)
```

- **Cliente:** Ao se conectar, o cliente recebe a resposta do servidor. Exemplo:

```
java
Copiar código
$ C:\tcp_exemplo.py cliente "127.0.0.1"
```

O cliente recebeu um nome de socket ('127.0.0.1', 57099)

4. Funções Importantes:

- **recv()**: Recebe dados à medida que ficam disponíveis.
- **sendall()**: Garante que todos os dados sejam transmitidos.
- **listen()**: Define o número de conexões pendentes (exemplo: `listen(1)` suporta uma conexão por vez).

5. Fechamento da Conexão:

- Após a comunicação, tanto o cliente quanto o servidor fecham o socket usando `close()`, o que finaliza a sessão TCP com o pacote FIN.

6. Opção SO_REUSEADDR:

- Define a opção `SO_REUSEADDR` para permitir que um socket seja vinculado a uma porta em uso por outro socket. Sem essa opção, ao tentar reexecutar o servidor, um erro ocorre:

```
makefile
Copiar código
OSError: [WinError 10048] Normalmente é permitida apenas uma
utilização de cada endereço de soquete.
```

- O `SO_REUSEADDR` permite que o socket seja forçado a vincular-se à porta imediatamente após a desconexão do cliente.

7. Estados de Espera do TCP:

- Após a desconexão, a conexão TCP pode permanecer por alguns minutos em estados **CLOSE-WAIT** e **TIME-WAIT**, durante os quais o sistema mantém o registro da conexão para garantir que os pacotes finais sejam entregues corretamente.

Este resumo abrange as etapas essenciais para a aplicação prática de sockets TCP, abordando as diferenças entre os tipos de sockets, os fluxos de comunicação e as configurações importantes para o funcionamento eficiente de um servidor TCP.

Clientes Promíscuos e Respostas Indesejadas em Sockets UDP

- **Códigos UDP inseguros:** No exemplo de código "udp_local.py", o cliente aceita qualquer resposta sem verificar a origem do datagrama, tornando-o vulnerável a respostas forjadas.
- **Risco de cliente promíscuo:** A função `recvfrom()` não valida o endereço de origem (IP + porta) do datagrama. Isso permite que um invasor envie uma resposta falsa que o cliente pode considerar legítima.
- **Simulação de ataque:** Interrompendo o servidor, é possível forjar uma resposta ao cliente antes do servidor, explorando a falha de segurança. O cliente então exibe a resposta forjada como se fosse autêntica.
- **Definição de cliente promíscuo:** Um cliente que aceita qualquer pacote sem validação é conhecido como cliente promíscuo. Ele é suscetível a aceitar dados de remetentes não autorizados.

- **Soluções:** Para evitar esse tipo de vulnerabilidade:
 - Usar criptografia para autenticar a comunicação.
 - Implementar protocolos com identificadores exclusivos nas respostas.
 - Verificar o endereço de origem do pacote de resposta.
 - Utilizar métodos que garantam conexões fim a fim.

Problemas na Transmissão com Redes e Confiabilidade do Socket UDP

- **Confiabilidade do UDP:** Em testes locais (loopback), não há perda de pacotes, mas em redes reais, a falta de confiabilidade do UDP pode levar à perda de pacotes, pois ele não garante entrega.
- **Protocolo IP:** Opera em "melhor esforço", sem garantias de tempo ou entrega completa dos pacotes.
- **TCP vs. UDP:**
 - **TCP:** Oferece confiabilidade e retransmissão de dados, porém adiciona latência, o que pode prejudicar multimídia em tempo real devido ao controle de congestionamento.
 - **UDP:** Mais adequado para transmissões contínuas (streaming), como áudio e vídeo, mantendo uma taxa constante de dados.
- **Simulação de perda de pacotes:**
 - O exemplo "udp_remote.py" utiliza uma função para responder aleatoriamente a apenas metade das requisições, simulando a perda de pacotes.
 - A interface "0.0.0.0" permite que o servidor escute em qualquer endereço de rede local, simulando um ambiente de rede mais real.
- **Objetivo:** Demonstrar a necessidade de implementar confiabilidade adicional ao cliente para lidar com perdas de pacotes em redes reais.

Tratamento de Perda de Dados em Aplicações UDP

- **Problemas de Confiabilidade do UDP:** O protocolo UDP não garante entrega de pacotes, exigindo que o cliente lide com possíveis perdas. Três cenários possíveis são:
 1. A resposta está atrasada.
 2. O pacote se perdeu.
 3. O servidor está fora do ar.
- **Estratégia de Tentativas Repetidas:**
 1. O cliente define um tempo de espera inicial (0,1 segundo no exemplo) com `settimeout()`. Se não houver resposta, o cliente retransmite a solicitação.
 2. A técnica de *backoff exponencial* é usada, dobrando o tempo de espera após cada tentativa. Esse método reduz o número de pacotes enviados em redes congestionadas, ajudando a recuperar a estabilidade.
- **Execução do Cliente:**
 1. Ao usar o exemplo `udp_remote.py`, o cliente simula perda de pacotes, enviando múltiplas solicitações se necessário, e o *backoff exponencial* é demonstrado.

2. O cliente verifica se as mensagens são descartadas e responde de acordo, até o ponto em que pode desistir após várias tentativas.

- **Cenários de Uso:**

1. Em aplicativos breves, é razoável desistir após tentativas fracassadas.
2. Em programas de longa duração, como um daemon de atualização, é útil estabelecer um tempo máximo de espera, como cinco minutos, para retomar tentativas após um longo período de desconexão.

Este resumo cobre o tratamento da confiabilidade de dados no UDP e a implementação prática de técnicas para compensar a falta de garantias de entrega.

Web Services e SOA

- **SOA (Arquitetura Orientada a Serviços):**

- Um padrão de arquitetura de software que utiliza princípios de computação distribuída.
- Funcionalidades do software são oferecidas como serviços acessíveis de forma independente.

- **Web Services:**

- Interface que permite comunicação entre diferentes sistemas (clientes e serviços, máquinas ou softwares).
- Facilitam o acesso a serviços em softwares construídos com arquitetura SOA, suportando linguagens e plataformas variadas.

- **Abordagens SOAP e REST:**

- **SOAP:** Protocolo estruturado para transmissão de dados em web services.
- **REST:** Alternativa ao SOAP, com enfoque em simplicidade e uso de métodos HTTP.

- **Objetivo:** O tema aborda os conceitos de web services, SOAP e REST, com exemplos práticos de uso em Python para ilustrar sua aplicação.

Web Services

- **Conceito:** Web services facilitam a comunicação e integração entre aplicações web, permitindo a interoperabilidade em sistemas distribuídos.

- **Arquitetura de Web Services:**

- Baseada nos modelos **SOAP** e **REST**:
 - **SOAP:** Formal, adequado para comunicação empresarial.
 - **REST:** Simples, ideal para interação com o consumidor final.

- **Evolução da Computação Distribuída:**

- Inicialmente, cliente e servidor se comunicavam apenas em redes internas.
- O uso de middlewares como **CORBA**, **DCOM** e **RMI** permitiu comunicação entre vários servidores.
- Com a internet, surgiram os web services, expandindo as chamadas remotas de métodos para a web.

- **Definição:** Um web service é uma interface que descreve operações acessíveis pela rede, permitindo transações e regras de negócio de uma aplicação serem acessadas por outras aplicações, independente da linguagem de programação ou sistema operacional.
- **Componentes da Arquitetura de Web Services:**
 - **Provedor de Serviços:** Cria, descreve e disponibiliza o web service.
 - **Consumidor de Serviços:** Utiliza o web service oferecido pelo provedor.
 - **Registro de Serviços:** Repositório onde serviços disponíveis são catalogados e encontrados.



Elementos da arquitetura Web services.

8 x 1

WSDL e UDDI, SOAP e REST

- **WSDL (Web Services Description Language):**
 - Linguagem baseada em XML para descrever web services.
 - Permite que clientes acessem as operações e assinaturas de um web service de forma automatizada.
- **UDDI (Universal Description, Discovery and Integration):**
 - Mecanismo para descoberta e publicação de web services.
 - Contém informações categorizadas sobre serviços e associa informações técnicas via WSDL.
- **SOAP e REST:**
 - **SOAP:** Padrão criado pelo W3C para garantir a interoperabilidade entre aplicações.
 - Baseado em XML e parte do padrão **WS-***, que inclui especificações como:
 - **WS-Addressing:** Define o transporte de mensagens.
 - **WS-Security:** Protocolo de segurança.
 - **REST:** Arquitetura alternativa que usa o protocolo HTTP.

- Simples e direta, voltada para o consumo de web services sem exigências de especificações formais como o SOAP.

REST (**Representational State Transfer**)

UDDI (*universal description, discovery and integration*)

WSDL (*web services description language*)

Modelo SOAP

- **SOAP (Simple Object Access Protocol):**
 - Protocolo XML para troca de informações em ambientes distribuídos, independente de sistema operacional, plataforma ou linguagem.
 - Utiliza o protocolo HTTP para encapsular chamadas e respostas de métodos web, podendo operar por RPC (chamadas remotas) ou troca de mensagens.
- **Modelos de Comunicação:**
 - **RPC:** Permite modelar chamadas de métodos com parâmetros e obter valores de retorno.
 - **Document:** Estrutura a mensagem em um formato de documento XML.
- **Estrutura da Mensagem SOAP:**
 - **Envelope:** Elemento principal e obrigatório que define o conteúdo da mensagem.
 - **Header:** Opcional; permite adicionar informações adicionais.
 - **Body:** Parte obrigatória que contém a informação a ser transmitida.
 - **Fault:** Elemento opcional dentro do Body para mensagens de erro/status.
- **Exemplo de Requisição e Resposta SOAP:**
 - A requisição XML invoca o método `GetModuloTema`, enviando `TemaNome` como parâmetro.
 - A resposta XML retorna módulos relacionados ao tema, exemplificando o uso de `GetModuloTemaResponse` no Body.

Esse resumo cobre os principais elementos do modelo SOAP, organizando sua estrutura e funções de forma clara para uma revisão eficaz.

Modelo REST

- **REST (Representational State Transfer):**
 - Conceito proposto por Roy Fielding em 2000, focado em utilizar os recursos do protocolo HTTP.
 - É uma arquitetura web, não um protocolo, e é mais simples que o SOAP.
- **Elementos do REST:**
 - **Cliente:** Quem solicita o serviço.
 - **Provedor:** Quem fornece o serviço.
 - **Protocolo HTTP:** Define o formato e transporte das mensagens.
- **Funcionamento:**

- O cliente inicia o processo enviando uma solicitação ao provedor.
- O provedor processa e responde ao cliente.
- O HTTP gerencia o formato e transporte das mensagens de requisição e resposta.
- **WADL:**
 - Em serviços RESTful, a **WADL (Web Application Description Language)** descreve os serviços de forma similar ao WSDL nos serviços SOAP.

Estrutura dos Recursos REST

- **URI Única para Recursos:**
 - Cada serviço REST corresponde a uma URI única, identificando recursos como "Tema" no exemplo.
 - Exemplos de URIs:
 - Listagem de todos os temas: `http://www.dominio.com.br/tema`
 - Inserção de um novo tema:
`http://www.dominio.com.br/tema/CreateTema/{nome-do-tema}`
- **Métodos HTTP no REST:**
 - REST utiliza métodos HTTP para manipulação de dados:
 - **GET:** Recupera ou lista recursos (ex: listar temas).
 - **POST:** Insere um novo recurso.
 - **PUT:** Edita um recurso existente.
 - **DELETE:** Exclui um recurso.
- **Resumo:** A estrutura REST segue a lógica dos métodos e códigos de retorno do HTTP, com cada operação correspondendo a uma ação específica sobre o recurso.

Códigos e Respostas REST

- **Códigos de Retorno HTTP:**
 - **200:** Sucesso na atualização do recurso.
 - **400 ou 404:** Indica erro na requisição.
- **Acesso a Recursos REST:**
 - Recursos REST podem ser acessados via URI, diretamente em navegadores ou por meio de um cliente programado para integrar aplicações.
 - Exemplo de URI para listar módulos de um tema específico:
`http://www.dominio.com.br/tema/GetModulosTema/Webservices`
- **Formato de Resposta:**
 - Respostas geralmente em JSON (padrão mais comum), mas REST suporta outros formatos de conteúdo:
 - **Tipos de Conteúdo:** `Application/xml`, `Application/json`, `Text/plain`, `Text/xml`, `Text/html`.
- **SOAP UI:**

- Ferramenta usada para testar web services SOAP e REST, destacando as diferenças entre ambos.

Protocolo SOAP com Python

- **Objetivo:** Codificar um web service SOAP e criar um cliente para consumir os serviços, utilizando Python. Através do aprendizado, será possível desenvolver web services simples e interagir com serviços de terceiros.
- **Estrutura da Aplicação:**
 - Utiliza o **Django** (framework Python) para hospedar o web service, definindo os recursos e compondo um cliente para consumir os serviços.
 - O **Spyne** é uma biblioteca usada para criar servidores RPC baseados em SOAP, com a vantagem de gerar automaticamente o arquivo **WSDL** para facilitar o acesso por terceiros.
 - A biblioteca **lxml** transforma XML para Python e vice-versa.
- **Instalação das Ferramentas:**
 - Usar **Miniconda** ou o console do **Spyder** para instalar os pacotes:
 - `pip install spyne`
 - `pip install lxml`
 - `pip install django`

Utilizando o "modelo resumo 2.3" resuma o texto a seguir:

Vou enviar o conteúdo de uma página de uma apostila de estudo. Preciso de um resumo claro e direto que mantenha as informações principais e mais relevantes. Ao resumir, siga estas instruções: Extraia as ideias centrais e informações essenciais. Mantenha o foco nos conceitos, definições, e fatos importantes que são necessários para uma revisão eficaz. Organize o resumo em frases simples e tópicos principais, se possível, facilitando a memorização e o entendimento. Seus resumos até agora tem sido um pouco vagos, por favor analisem melhor o texto e não descarte dados importantes. Texto a ser resumido:

Vou enviar o conteúdo de uma página de uma apostila de estudo. Preciso de um resumo claro e direto que mantenha as informações principais e mais relevantes. Ao resumir, siga estas instruções: Extraia as ideias centrais e informações essenciais. Mantenha o foco nos conceitos, definições, e fatos importantes que são necessários para uma revisão eficaz. Eu não quero que você simplifique as coisas ao ponto de deixar o mínimo de informações possível, não é esse o objetivo aqui, o objetivo é pegar o texto original, e simplificá-lo, tirando quaisquer redundâncias, repetições, ou palavras desnecessárias. Se algo pode ser dito eficientemente com 5 palavras, não há necessidade de usar 10. Texto a ser resumido:

Eu não quero que você simplifique as coisas ao ponto de deixar o mínimo de informações possível, não é esse o objetivo aqui, o objetivo é pegar o texto original, e simplificá-lo, tirando quaisquer

redundâncias, repetições, ou palavras desnecessárias. Se algo pode ser dito eficientemente com 5 palavras, não há necessidade de usar 10.

Núcleo da rede:

Múltiplos roteadores interconectados.

Formas de transferir dados na rede:

computação do circuito: usa um canal dedicado para cada conexão

ex: dirigir com uma rota fixa

comunicação de pacotes: dados são enviados em “blocos” discretos

ex: dirigir com mudança de rotas dinamicamente

Comutação de circuitos: FDM e TDM

Redes datagrama

Redes de circuitos virtuais

Fonte de atraso de pacotes: processamento nos nós, enfileiramento

Atraso de transmissão:

L = tamanho do pacote/bits

R = largura de banda do link/bps)

Tempo para enviar bits ao link = L/R

Tempo para empurrar todos os pacotes no enlace.

Atraso de propagação:

d = comprimento do link físico

s = velocidade de propagação no meio ($\sim 2 \times 10^8$ m/s)

Atraso de propagação = d/s

Tempo para o pacote transitar no enlace

Construir um sistema complexo é difícil.

Abordagem: “dividir e conquistar” (divide o meu trabalho em partes pequenas ou camadas)

Modularização:

Ideia básica: cada etapa depende da anterior mas não é necessário saber como a etapa anterior foi realizada.

Analogia: linha de produção de carros.

Modelo ISO/OSI

ISO International standards organization

OSI Open Systems Interconnection

É um padrão e não uma implementação.

Não há uma implementação dele.

O Bluetooth, zigbee e a internet seguem esse padrão.

Camada de aplicação, camada de apresentação, camada de sessão, camada de transporte, camada de rede, camada de enlace, camada física.

Encapsulamento de dados: pegar o dado da camada anterior e colocá-lo na camada seguinte.

Modelo ISO/OSI versus TCP/IP

ISO/OSI -> classe

TCP/IP -> objeto (Camada de aplicação, camada de transporte, camada de rede, camada de acesso ao meio)

TCP/IP é a verdadeiro modelo da internet.

Exemplos de aplicações de rede: e-mail, mensagem instantânea, login remoto, telefonia via internet, etc.

Arquiteturas de aplicação:

Cliente-servidor

Peer-to-peer (P2P)

Híbrida de cliente-servidor e P2P

Arquitetura de aplicação é diferente de arquitetura de rede.

FTP file transfer protocol

SMTP simple message transfer protocol

Multiplexação e demultiplexação

IETF internet engineering task force

RFC 768

RFC = request for comments

UDP = user datagram protocol

Soma de verificação UDP

Checksum

RTT round trip time