Resumo Redes

Resumo tópicos da Prova 1

Abril 2025
24.04.2025

INFORMAÇÕES
Baseado nos materiais de aula
Cópia livre para a circulação

AUTOR
Enzo Serenato
enzoserenato@gmail.com
Ponta Grossa, Paraná

Sumário

1. Comutação

1.1. Comutação de circuitos

A comutação de circuitos é um método de comunicação em redes no qual um caminho dedicado é estabelecido entre os dispositivos antes da transmissão de dados. Esse caminho permanece reservado exclusivamente para a comunicação até que a transmissão seja concluída. Em redes telefônicas analógicas tradicionais, um circuito físico é mantido ativo durante toda a conversa, independentemente de haver transmissão contínua de dados. O canal não é compartilhado com outros usuários.

Vantagens:

- Garantia de largura de banda.
- Baixa latência.

Desvantagem:

- Ineficiência no uso de recursos, pois o circuito permanece reservado mesmo quando não há transmissão de dados.
- Requer softwares complexos de sinalização para coordenar comutadores ao longo do caminho.

Multiplexação:

- **FDM (Multiplexação por Divisão de Frequência)**: Divide o espectro de frequência do enlace em bandas, alocando uma banda para cada conexão (ex.: 4 kHz em redes telefônicas).
- **TDM (Multiplexação por Divisão de Tempo)**: Divide o tempo em quadros fixos com compartimentos (slots), reservando um slot por quadro para cada conexão. A taxa de transmissão de um circuito é a taxa do quadro vezes o número de bits por slot (ex.: 8 mil quadros/s com slots de 8 bits = 64 kbits/s).

Exemplo numérico:

- Para enviar um arquivo de 640 kbits por uma rede TDM com 24 slots e taxa de enlace de 1,536 Mbits/s, cada circuito tem taxa de 64 kbits/s. O tempo de transmissão é 10 segundos, mais 500 ms para ativação do circuito, totalizando 10,5 segundos, independente do número de enlaces.

1.2. Comutação de pacotes

A comutação de pacotes, inventada nos anos 1960, consiste em dividir mensagens em pequenos blocos chamados pacotes, que contêm a identificação do destinatário. Vários transmissores compartilham os enlaces da rede, e os roteadores encaminham os pacotes aos destinatários. Computadores ociosos não ocupam recursos da rede, promovendo um compartilhamento eficiente.

Vantagens:

- Compartilhamento eficiente dos recursos da rede.
- Flexibilidade no uso da infraestrutura.
- Permite suportar mais usuários que a comutação de circuitos, especialmente em tráfego esporádico (ex.: 35 usuários com 10% de atividade podem compartilhar um enlace de 1 Mbit/s com probabilidade de congestionamento de apenas 0,0004).

Desvantagens:

- Congestionamentos podem aumentar a latência.

- Possibilidade de perda de pacotes.

Transmissão armazena-e-reenvia:

- Comutadores de pacotes recebem o pacote inteiro antes de transmiti-lo ao próximo enlace, causando atraso de transmissão (ex.: pacote de L bits em enlace de R bits/s leva L/R segundos).
- Para N enlaces, o atraso fim a fim de um pacote é N **(L/R). Para P pacotes, o atraso total é (N + P - 1)** (L/R).

Atrasos de fila e perda:

- Pacotes esperam em buffers de saída se o enlace estiver ocupado, causando atrasos de fila variáveis.
- Buffers finitos podem levar à perda de pacotes se lotados.

	Circuitos	**Pacotes**
Linha	Caminho dedicado, reservado exclusivamente	Enlaces compartilhados por pacotes
Recursos	Ineficiente, reservado mesmo sem uso	Eficiente, usado sob demanda
Latência	Baixa, garantida	Variável, sujeita a congestionamentos
Perda	Nenhuma, conexão estável	Possível devido a congestionamentos

2. Protocolo de comunicação de dados

Um protocolo de comunicação define o formato, a ordem das mensagens trocadas e as ações a serem tomadas pelas entidades, incluindo o tratamento de erros. Ele especifica desde detalhes de baixo nível, como voltagem na transmissão, até questões de alto nível, como o formato e a semântica das mensagens de uma aplicação. Protocolos são organizados em uma pilha de camadas, onde cada camada tem uma função distinta, e as camadas correspondentes entre dispositivos se comunicam. A interoperabilidade é garantida por padrões, evitando dependência de fabricantes ou provedores.

Pilha de protocolos, cabeçalhos e encapsulamento: Os dados passam por várias camadas, onde cada uma adiciona um cabeçalho com informações específicas (encapsulamento). Por exemplo:

- **Camada de Aplicação**: Gera a mensagem (ex.: HTML).
- **Camada de Transporte**: Adiciona cabeçalho TCP (números de porta, sequência).
- **Camada de Rede**: Adiciona cabeçalho IP (endereços IP, TTL).
- **Camada de Enlace**: Adiciona cabeçalho Ethernet (endereços físicos, CRC).

O processo inverso ocorre no destino, onde cada camada remove seu cabeçalho.

Vantagens da arquitetura em camadas:

- Modularidade: Facilita modificações em uma camada sem afetar as demais, desde que o serviço oferecido permaneça o mesmo.

- Simplificação: Permite discutir e projetar partes específicas de um sistema complexo.

Desvantagens:
- Possível duplicação de funcionalidades entre camadas (ex.: recuperação de erros no enlace e fim a fim).
- Dependência de informações de outras camadas, violando a separação (ex.: necessidade de carimbos de tempo).

3. Modelo OSI

O Modelo OSI (Open Systems Interconnection) é um padrão que organiza a comunicação em redes em sete camadas, cada uma com funções específicas:

1. **Física**: Define o meio de transmissão e propriedades elétricas (ex.: frequências, sinais).
2. **Enlace de Dados**: Gerencia a comunicação com a rede física, incluindo endereçamento físico e acesso ao meio.
3. **Rede**: Define a comunicação entre redes interconectadas, como endereçamento IP.variable and unpredictable end-to-end delays due to variable and unpredictable queuing delays).
4. **Transporte**: Garante comunicação confiável entre processos, com controle de fluxo e congestionamento.
5. **Sessão**: Gerencia sessões entre aplicações, incluindo delimitação e sincronização da troca de dados.
6. **Apresentação**: Trata da formatação, compressão, codificação e criptografia dos dados.
7. **Aplicação**: Define como aplicações se comunicam (ex.: HTTP, SMTP).

Comparação com a pilha da Internet:
- A pilha da Internet tem cinco camadas: Física, Enlace, Rede, Transporte e Aplicação.
- As funções das camadas de Sessão e Apresentação do OSI são incorporadas na camada de Aplicação da Internet, deixando ao desenvolvedor a responsabilidade de implementar esses serviços, se necessário.

Vantagens dos padrões:
- Interoperabilidade entre sistemas de diferentes fabricantes.
- Facilita o desenvolvimento e manutenção de redes.

Desvantagens:
- Complexidade adicional devido à padronização.
- Possível rigidez para inovações fora do padrão.

3.1. Unidade de dados de protocolo (PDU)

As PDUs são as unidades de dados trocadas em cada camada da pilha de protocolos:
- **Camada de Aplicação**: Mensagem (ex.: HTML).
- **Camada de Transporte**: Segmento (ex.: TCP com cabeçalho).
- **Camada de Rede**: Pacote ou Datagrama (ex.: IP com cabeçalho).
- **Camada de Enlace**: Quadro (ex.: Ethernet com cabeçalho).

3.2. História do surgimento da Internet e dos protocolos TCP e IP

A Internet surgiu da ARPANET, uma rede dos anos 1960 criada para compartilhar recursos computacionais caros, utilizando comutação de pacotes. Leonard Kleinrock, Paul Baran e Donald Davies contribuíram com conceitos fundamentais, como teoria de filas e resistência a falhas. Nos anos 1970, Robert Kahn e Vinton Cerf desenvolveram o TCP/IP, permitindo a interconexão de redes heterogêneas. A virtualização do endereçamento IP tornou a Internet independente da tecnologia de rede subjacente. O TCP/IP ganhou popularidade com sua implementação no Unix BSD, SunOS, e pela criação da NSFNET (1987), além de empresas como Cisco e Microsoft.

3.2.1. ARPANET e NSFNet

- **ARPANET**: Iniciada em 1969, foi a primeira rede a usar comutação de pacotes, conectando universidades e centros de pesquisa. Em 1971, tinha poucos nós; em 1980, expandiu-se significativamente.
- **NSFNET**: Criada em 1987, conectou instituições acadêmicas e impulsionou a Internet comercial, adotando TCP/IP como padrão.

3.3. Provedores de Internet (ISPs)

ISPs fornecem conectividade à Internet, organizados em uma hierarquia:

- **ISPs de Nível 1**: Formam o núcleo da Internet, interconectando-se globalmente sem pagar por tráfego.
- **ISPs de Níveis Inferiores**: Conectam-se a ISPs de nível 1, pagando pelo tráfego. Podem ser regionais ou locais.
- **Sistemas Autônomos (AS)**: Redes independentes com políticas próprias de roteamento, operadas por ISPs ou grandes provedores de conteúdo (ex.: Google).

Estrutura da Internet:

- **Pontos de Presença (PoPs)**: Grupos de roteadores onde ISPs clientes se conectam a ISPs provedores.
- **Multi-homing**: ISPs conectam-se a múltiplos provedores para maior confiabilidade.
- **Emparelhamento**: ISPs no mesmo nível conectam-se diretamente para reduzir custos, sem taxas.
- **Pontos de Troca da Internet (IXPs)**: Locais onde vários ISPs se emparelham.
- **Redes de Provedores de Conteúdo**: Ex.: Google, com redes privadas que se conectam diretamente a ISPs de níveis inferiores ou IXPs, reduzindo custos e controlando a entrega de serviços.

4. Tipos de redes

- **Rede de Acesso**: Conecta usuários finais (ex.: residências, empresas) ao núcleo da rede, via ISPs.
- **Núcleo da Rede**: Composto por ISPs de nível 1 e roteadores interconectados, gerencia o tráfego global.

Classificação das redes:

- **Quanto ao tamanho**: Domésticas/SOHO, porte médio, grandes corporativas.
- **Quanto à abrangência**:
  - LAN (Local): Ex.: redes de escritórios.
  - MAN (Metropolitana): Ex.: redes de cidades.
  - WAN (Longa Distância): Ex.: Internet.

    - PAN (Pessoal): Ex.: Bluetooth.
    - SAN (Armazenamento): Ex.: redes de data centers.
- **Quanto à disponibilidade**:
    - Pública: Oferecida por ISPs a assinantes.
    - Privada: Uso exclusivo de empresas.
    - Governamental: Ex.: RNP, que conecta instituições de ensino e pesquisa no Brasil.

Tecnologias de Rede de Acesso:

- **Fixas:**
    - **ADSL**: Usa par metálico telefônico, com multiplexação por divisão de frequência (downstream: 50 kHz a 1 MHz; upstream: 4 kHz a 50 kHz; voz: 0 a 4 kHz). Velocidade máxima de 52 Mbits/s downstream, mas upstream limitado.
    - **HFC (Híbrida Fibra-Coaxial)**: Combina fibra óptica até um ponto de distribuição e cabo coaxial até o usuário, usando protocolo DOCSIS. Oferece até 1,2 Gbits/s compartilhados, requerendo protocolos de acesso múltiplo.
    - **FTTx**: Fibra óptica, como FTTH (fibra até a residência).
    - **Wi-Fi Comunitário/Redes Mesh**: Extensões de redes fixas para cobertura urbana/rural.
- **Sem Fio:**
    - **Celular**: 4G, 5G (móvel e fixo).
    - **WiMax**: Acesso sem fio de longo alcance, usado em áreas rurais.
    - **Satélite**: Para áreas remotas, com maior latência em sistemas geoestacionários e melhor desempenho em órbitas baixas (LEO).

Enlaces de Dados:

- **Meios Guiados:**
    - **Par Trançado**: Categoria 5 (100 Mbits/s, 1 Gbit/s), Categoria 6 (10 Gbits/s).
    - **Cabo Coaxial**: Usado em HFC.
    - **Fibra Ótica**: Alta capacidade, usada em FTTx.
- **Meios Não Guiados:**
    - **Rádio**: Wi-Fi, 4G/5G, WiMax, Bluetooth, micro-ondas terrestres, satélites.

5. ICMP, Ping & traceroute

- **ICMP (Internet Control Message Protocol)**: Usado para mensagens de controle e diagnóstico na camada de rede (ex.: erros, ecos). É encapsulado em pacotes IP.
- **Ping**: Usa ICMP para testar a conectividade entre dois hosts, enviando pacotes de eco e medindo o tempo de resposta (latência). Resultados mostram se o destino é alcançável e a qualidade da conexão.
- **Traceroute**: Diagnostica a rota de pacotes, enviando pacotes com TTL crescente para identificar roteadores no caminho. Cada roteador retorna uma mensagem ICMP, permitindo medir atrasos de ida e volta e reconstruir a rota. Resultados incluem o número do roteador, nome, endereço IP e tempos de resposta.

6. Protocolos e governança na Internet

A governança da Internet envolve protocolos, interoperabilidade, segurança e leis. Organizações padronizam protocolos e gerenciam recursos como endereços IP e nomes de domínio.

6.1. Órgãos padronizadores

- **IETF (Internet Engineering Task Force)**: Define protocolos da Internet (ex.: TCP, IP, HTTP) via RFCs.
- **IEEE**: Padroniza camadas física e de enlace (ex.: Ethernet, Wi-Fi).
- **W3C**: Define padrões para a Web (ex.: HTML, CSS).
- **ITU, ISO, ANSI, ITIC, EIA, TIA**: Contribuem com padrões técnicos globais.
- **ICANN**: Gerencia políticas da Internet, incluindo a IANA, que coordena DNS, endereços IP, portas e fusos horários.
- **RIRs**: Distribuem endereços IP regionalmente (ex.: LACNIC para América Latina).
- **CGI.br**: No Brasil, gerencia domínios (.br), segurança (Cert.br), e promove padrões Web (W3C Brasil).

6.2. IETF e RFCs

A IETF, criada em 1986, publica RFCs, documentos que descrevem protocolos, procedimentos e informações da Internet. Exemplos:

- RFC 791: Define o protocolo IP.
- RFC 1180: Explica TCP/IP didaticamente.
- Protocolos em RFCs: TCP, UDP, HTTP, SMTP, ICMP, DNS, entre outros.

7. Atrasos

Os atrasos em redes de comutação de pacotes afetam o desempenho de aplicações. O atraso nodal total é a soma de quatro componentes: processamento, fila, transmissão e propagação.

7.1. Atraso de transmissão

Tempo necessário para transmitir todos os bits de um pacote para o enlace, calculado como

$$\frac{L}{R}$$

, onde L é o tamanho do pacote (bits) e R é a taxa de transmissão do enlace (bits/s). Exemplo: Para um pacote de 10 Mbits em um enlace de 100 Mbits/s, o atraso é 0,1 segundo. Varia de micro a milissegundos.

7.2. Atraso de propagação

Tempo para um bit se propagar do início ao fim do enlace, calculado como

$$\frac{d}{s}$$

, onde d é a distância e s é a velocidade de propagação (2 a 3 × 10^8 m/s). Em WANs, é da ordem de milissegundos. Depende da distância, não do tamanho do pacote.

7.3. Atraso de filas

Tempo que um pacote espera na fila antes da transmissão, dependendo do número de pacotes à frente. Varia de zero (fila vazia) a significativo (tráfego intenso). É o componente mais complexo, influenciado pela intensidade de tráfego

$$\frac{L_a}{R}$$

. Quando a intensidade se aproxima de 1, o atraso cresce rapidamente. Filas finitas podem causar perda de pacotes.

7.4. Atraso de processamento

Tempo para examinar o cabeçalho do pacote, verificar erros e determinar o enlace de saída. Geralmente na ordem de microsegundos, é quase sempre desprezível, mas afeta a produtividade máxima do roteador.

8. Camada de Aplicação

A camada de aplicação é a camada mais alta da arquitetura de redes, responsável por fornecer serviços diretamente às aplicações dos usuários. Ela permite que programas em sistemas finais se comuniquem por meio da rede, utilizando protocolos específicos para estruturar a troca de mensagens. As aplicações de rede, como a Web, e-mail, transferência de arquivos e VoIP, são a razão principal da existência das redes de computadores, impulsionando o desenvolvimento de protocolos para suportar suas necessidades.

8.1. Requisitos de Redes das Aplicações

As aplicações de rede possuem diferentes requisitos em relação a **perda de dados**, **vazão** (throughput) e **atraso** (temporização). Esses requisitos determinam qual protocolo de transporte (TCP ou UDP) é mais adequado para cada aplicação. Abaixo estão as principais dimensões:

- **Perda de Dados**:
  - Aplicações como transferência de arquivos, e-mail e documentos Web exigem **transferência confiável** sem perda de dados, pois qualquer perda pode comprometer a integridade da informação.
  - Aplicações multimídia, como streaming de vídeo ou telefonia por IP, são **tolerantes à perda**, pois pequenas perdas não afetam significativamente a experiência do usuário.
- **Vazão (Throughput)**:
  - **Aplicações sensíveis à largura de banda**, como videoconferência e VoIP, requerem uma vazão mínima garantida (ex.: 32 kbits/s para voz, até 5 Mbits/s para vídeo).
  - **Aplicações elásticas**, como e-mail e transferência de arquivos, podem se adaptar à vazão disponível, embora maior vazão melhore o desempenho.
- **Atraso (Temporização)**:
  - Aplicações em tempo real, como jogos interativos e telefonia por IP, são sensíveis ao atraso, exigindo tempos de resposta na ordem de décimos de segundo.
  - Aplicações não interativas, como e-mail e transferência de arquivos, não possuem restrições estritas de atraso.

Aplicação	Perda de Dados	Vazão	Sensibilidade ao Tempo
Transferência de Arquivos	Sem perda	Elástica	Não

Aplicação	Perda de Dados	Vazão	Sensibilidade ao Tempo
E-mail	Sem perda	Elástica	Não
Web (Documentos)	Sem perda	Elástica (alguns kbits/s)	Não
Telefonia/Videoconferência	Tolerante à perda	Áudio: 10 kbits/s - 1 Mbit/s Vídeo: 10 kbits/s - 5 Mbits/s	Sim (décimos de segundo)
Streaming de Vídeo	Tolerante à perda	Vídeo: 10 kbits/s - 5 Mbits/s	Sim (alguns segundos)
Jogos Interativos	Tolerante à perda	Poucos kbits/s - 10 kbits/s	Sim (décimos de segundo)
Mensagem Instantânea	Sem perda	Elástica	Sim e não

8.2. Arquiteturas de Aplicações

As arquiteturas de aplicações definem como os programas são organizados nos sistemas finais para realizar a comunicação. Existem duas arquiteturas predominantes: **Cliente/Servidor** e **Peer-to-Peer (P2P)**.

8.2.1. Arquitetura Cliente/Servidor

Na arquitetura cliente/servidor, um **servidor** está sempre ativo, aguardando requisições de múltiplos **clientes**. O servidor possui um endereço IP fixo e é geralmente hospedado em datacenters para suportar alta demanda.

- **Características**:
  - O servidor atende requisições de clientes sem comunicação direta entre os clientes.
  - Exemplos: Web (HTTP), e-mail (SMTP, IMAP, POP3), FTP, e servidores de banco de dados.
  - Escalabilidade pode ser um desafio, exigindo datacenters com milhares de servidores para serviços populares (ex.: Google, Amazon).
- **Vantagens**:
  - Centralização facilita gerenciamento e manutenção.
  - Alta confiabilidade devido à infraestrutura dedicada.
- **Desvantagens**:
  - Alto custo de infraestrutura (energia, largura de banda).
  - Ponto único de falha se o servidor não for redundante.

8.2.2. Arquitetura P2P

Na arquitetura P2P, não há um servidor central dedicado. Os **pares** (peers) se comunicam diretamente, atuando simultaneamente como clientes e servidores.

- **Características**:
  - Cada par contribui com recursos (largura de banda, armazenamento), tornando a arquitetura autoescalável.
  - Exemplos: BitTorrent, Skype (para chamadas), sistemas de compartilhamento de arquivos.
  - Comunicação direta entre pares, sem passar por servidores intermediários.
- **Vantagens**:
  - Baixo custo, pois não requer infraestrutura centralizada.
  - Autoescalabilidade: mais pares aumentam a capacidade do sistema.
- **Desvantagens**:
  - Desafios de segurança devido à natureza distribuída.
  - Necessidade de incentivos para que usuários compartilhem recursos.
  - Impacto em ISPs residenciais devido ao tráfego simétrico.

8.2.3. Arquiteturas Híbridas

Algumas aplicações combinam elementos cliente/servidor e P2P. Por exemplo, em mensagens instantâneas, servidores rastreiam endereços IP, mas as mensagens são trocadas diretamente entre pares. Proxies web também são híbridos, atuando como servidores para usuários finais e clientes para servidores de destino.

8.3. Modelos de Comunicação

As aplicações utilizam dois principais modelos de comunicação fornecidos pela camada de transporte: **Fluxo de Bytes** (TCP) e **Troca de Mensagens** (UDP).

8.3.1. Fluxo de Bytes (TCP)

O modelo de fluxo de bytes, implementado pelo protocolo TCP, é orientado à conexão e oferece comunicação confiável.

- **Características**:
  - **Conexão**: Uma conexão full-duplex é estabelecida antes da troca de dados, com handshake inicial e encerramento explícito.
  - **Confiabilidade**: Garante entrega de dados sem perdas, duplicações ou erros, na ordem correta.
  - **Controle de Congestionamento**: Limita a taxa de envio em redes congestionadas.
  - **Flexibilidade**: Dados são enviados como uma sequência de bytes, sem estrutura fixa, sendo responsabilidade da aplicação interpretar o conteúdo.
  - **Aplicações**: E-mail (SMTP), Web (HTTP), transferência de arquivos (FTP), acesso remoto (Telnet, SSH).
- **Funcionamento**:
  - O cliente inicia a conexão com o servidor (via `connect()`).
  - O servidor aguarda conexões (via `accept()`).
  - Dados são enviados em blocos, mas recebidos como fluxo contínuo.

8.3.2. Troca de Mensagens (UDP)

O modelo de troca de mensagens, implementado pelo protocolo UDP, é sem conexão e oferece um serviço de "melhor esforço".

- **Características**:
  - **Sem Conexão**: Não há handshake ou encerramento; mensagens são enviadas diretamente.
  - **Não Confiável**: Pode haver perda, duplicação ou desordem de mensagens.
  - **Tamanho Fixo**: Mensagens (datagramas) têm tamanho limitado (até 64 KB).
  - **Baixa Latência**: Ideal para aplicações sensíveis ao tempo, como streaming e VoIP.
  - **Aplicações**: Telefonia por IP (RTP), jogos interativos, streaming de vídeo (em alguns casos).
- **Funcionamento**:
  - O cliente envia uma mensagem ao servidor, que responde sem estabelecer conexão.
  - Mensagens enviadas e recebidas mantêm sua integridade (tamanho e conteúdo).

8.3.3. Outros Protocolos de Transporte

Além de TCP e UDP, outros protocolos são relevantes para a camada de aplicação:

- **TLS (Transport Layer Security)**: Fornece autenticação, criptografia e integridade para aplicações seguras (ex.: HTTPS).
- **RTP/RTCP**: Usados em aplicações multimídia em tempo real, como streaming e VoIP.
- **SCTP**: Alternativa ao TCP com maior flexibilidade, mas pouco utilizado.

8.3.4. Escolha entre TCP e UDP

A escolha do protocolo depende dos requisitos da aplicação:

- **TCP**: Preferido para aplicações que exigem confiabilidade (ex.: e-mail, Web, FTP).
- **UDP**: Usado em aplicações tolerantes à perda e sensíveis ao tempo (ex.: VoIP, streaming).
- **Exemplo**: Aplicações como Skype usam UDP para baixa latência, mas podem recorrer ao TCP se o UDP for bloqueado por firewalls.

8.4. Conceitos de Clientes e Servidores

Os termos "cliente" e "servidor" têm diferentes significados dependendo do contexto: conexões TCP, programas ou hardware.

8.4.1. Em Relação a Conexões TCP

- **Cliente**: Inicia a conexão com o servidor (via `connect()`). Precisa conhecer o endereço IP e a porta do servidor.
- **Servidor**: Aguarda conexões de clientes (via `accept()`). Escuta em uma porta específica.
- **Comunicação**: Bidirecional (full-duplex), persistente até o encerramento explícito.

8.4.2. Em Relação a Programas

- **Programas Clientes**:
  - Iniciam após o servidor estar ativo.

    - Executados temporariamente por usuários, geralmente em dispositivos locais.
    - Iniciam a comunicação com o servidor.
    - Ex.: Navegadores web, clientes de e-mail.
- **Programas Servidores**:
    - Iniciam com o sistema e permanecem ativos.
    - Aguardam conexões passivamente, atendendo múltiplos clientes simultaneamente (via threads ou eventos de E/S).
    - Podem requerer hardware robusto para alta demanda.
    - Ex.: Servidores web (Apache), servidores de e-mail.
- **Híbridos**: Programas podem ser clientes e servidores ao mesmo tempo (ex.: proxies web, sistemas P2P).

8.4.3. Em Relação ao Hardware

- **Clientes**: Geralmente dispositivos de usuário (PCs, smartphones) com hardware padrão.
- **Servidores**: Computadores de alta capacidade, muitas vezes em datacenters, projetados para lidar com múltiplas conexões simultâneas.

8.4.4. Identificação de Servidores

Clientes identificam servidores por:

- **Endereço IP**: Identifica o computador hospedeiro (obtido via DNS).
- **Porta**: Especifica o serviço (ex.: 80 para HTTP, 25 para SMTP).

8.5. Middleware de Comunicação

O **middleware** é uma camada de software entre a aplicação e o sistema operacional, usada em sistemas distribuídos para simplificar o desenvolvimento.

- **Funções**:
    - Oculta complexidade de redes e sistemas heterogêneos.
    - Fornece interfaces padronizadas para interoperabilidade e portabilidade.
    - Reduz duplicação de esforços com serviços comuns.
- **Exemplos**:
    - RPC (Chamada Remota de Procedimentos).
    - Objetos Distribuídos (ex.: CORBA).
    - ZeroMQ (mensagens distribuídas).
    - Web Services.
    - MPI (computação paralela).

8.6. Exemplos de Aplicações

Algumas aplicações populares ilustram os conceitos da camada de aplicação:

- **Web (HTTP)**: Usa TCP, arquitetura cliente/servidor, com navegadores como clientes e servidores web (ex.: Apache) como servidores.
- **E-mail (SMTP, IMAP, POP3)**: Usa TCP, com servidores de correio e clientes de e-mail; SMTP também permite comunicação servidor-servidor.
- **Transferência de Arquivos (FTP)**: Usa TCP, com servidores de arquivos e clientes FTP.

- **DNS**: Serviço de diretório que mapeia nomes de domínio para endereços IP, essencial para outras aplicações.
- **P2P (BitTorrent)**: Usa UDP ou TCP, com pares atuando como clientes e servidores simultaneamente.

8.7. Protocolos de Camada de Aplicação

Os **protocolos de camada de aplicação** definem como as mensagens são estruturadas e trocadas entre processos. Eles especificam:

- Tipos de mensagens (requisição/resposta).
- Sintaxe e semântica dos campos.
- Regras para envio e resposta.
- **Exemplos**:
  - **HTTP**: Protocolo da Web, baseado em requisições e respostas.
  - **SMTP**: Protocolo para envio de e-mails.
  - **DNS**: Protocolo para resolução de nomes.
  - **RTP**: Protocolo para multimídia em tempo real.
- **Públicos vs. Proprietários**: Protocolos como HTTP e SMTP são públicos (definidos em RFCs), enquanto outros, como o do Skype, são proprietários.

8.8. Segurança na Camada de Aplicação

A segurança é um requisito crítico para muitas aplicações. Embora TCP e UDP não ofereçam criptografia nativa, o protocolo **SSL/TLS** pode ser integrado na camada de aplicação para:

- **Criptografia**: Proteger a confidencialidade dos dados.
- **Integridade**: Garantir que os dados não foram alterados.
- **Autenticação**: Verificar a identidade das partes.

Exemplo: HTTPS usa SSL/TLS sobre TCP para comunicações seguras na Web.

8.9. Desafios e Tendências

- **Escalabilidade**: Arquiteturas cliente/servidor exigem datacenters robustos, enquanto P2P depende da colaboração de pares.
- **Segurança**: Aplicações P2P enfrentam desafios devido à sua natureza distribuída.
- **Amigabilidade com ISPs**: Aplicações P2P precisam minimizar o impacto em redes residenciais.
- **Novas Aplicações**: A Internet continua a ver o surgimento de aplicações inovadoras, como redes sociais, jogos online e streaming.

9. Sistema de Nomes de Domínio (DNS)

O Sistema de Nomes de Domínio (DNS) é um protocolo de camada de aplicação essencial para a Internet, funcionando como um banco de dados distribuído que traduz nomes de hospedeiros (como www.exemplo.com) em endereços IP, permitindo a comunicação entre dispositivos. Ele opera no modelo cliente-servidor, utilizando o protocolo UDP na porta 53, e é projetado para ser tolerante a falhas, robusto contra ataques e eficiente. Além da tradução de nomes, o DNS suporta funções como mapeamento de apelidos (aliasing), distribuição de carga entre servidores replicados e gerenciamento de registros de servidores de correio.

9.1. Servidores recursivos e autoritativos

O DNS é composto por uma hierarquia de servidores, dividida em três classes principais: servidores raiz, servidores de domínio de alto nível (TLD) e servidores autoritativos. Além desses, os servidores DNS recursivos (ou resolvers) desempenham um papel crucial.

- **Servidores autoritativos**: Contêm os registros originais de um domínio específico, como example.com.

São responsáveis por armazenar e fornecer informações como endereços IP (registros A ou AAAA) ou servidores de correio (registros MX). Alterações nos registros de um domínio são feitas nesses servidores. Por exemplo, uma universidade como a UEPG pode manter um servidor autoritativo para o domínio uepg.br.

- **Servidores recursivos**: Atuam como intermediários entre os clientes (como navegadores) e a hierarquia de servidores DNS.

Quando um cliente envia uma consulta recursiva, o servidor recursivo realiza consultas iterativas, começando, se necessário, pelos servidores raiz, descendo até os servidores autoritativos para obter a resposta. Normalmente, os clientes conhecem pelo menos um servidor recursivo local, configurado manualmente ou via DHCP/NDP.

9.2. Funcionamento das consultas DNS

As consultas DNS podem ser recursivas ou iterativas:

- **Consultas recursivas**: O cliente solicita que o servidor recursivo obtenha a resposta completa, delegando a responsabilidade de navegar pela hierarquia DNS.

Por exemplo, um navegador consultando www.amazon.com envia uma consulta recursiva ao servidor DNS local.

- **Consultas iterativas**: O servidor retorna apenas a próxima etapa da resolução, como o endereço de um servidor TLD ou autoritativo, e o cliente (ou outro servidor) continua o processo.

Na prática, a consulta inicial do cliente ao servidor local é recursiva, enquanto as consultas subsequentes na hierarquia são iterativas.

Um exemplo de resolução: para traduzir www.amazon.com, o cliente contata o servidor DNS local, que consulta um servidor raiz. O servidor raiz retorna o endereço de um servidor TLD para .com, que, por sua vez, indica o servidor autoritativo para amazon.com. Finalmente, o servidor autoritativo fornece o endereço IP de www.amazon.com. Esse processo pode envolver até oito mensagens DNS (quatro consultas e quatro respostas).

9.3. Conteúdo dos registros DNS

Os servidores DNS armazenam registros de recursos (RR), que são tuplas com quatro campos: (Nome, Valor, Tipo, TTL). O TTL (Time to Live) determina por quanto tempo o registro pode ser armazenado em cache. O significado de Nome e Valor depende do tipo de registro:

- **Nome**: Geralmente o nome do domínio ou hospedeiro (ex.: www.exemplo.com).
- **Valor**: O dado associado, como um endereço IP ou nome de servidor.
- **Tipo**: Define a função do registro (ex.: A, MX, NS).
- **TTL**: Tempo em segundos que o registro permanece válido em cache.

As mensagens DNS, sejam de consulta ou resposta, seguem um formato com seções: cabeçalho (contendo identificadores e flags), pergunta (nome e tipo consultados), resposta (registros retornados), autoridade e adicional (informações complementares).

9.4. Principais tipos de registros

Os tipos de registros mais comuns incluem:

- **A**: Mapeia um nome de hospedeiro para um endereço IPv4. Exemplo: (www.exemplo.com, 192.0.2.1, A).
- **AAAA**: Mapeia um nome de hospedeiro para um endereço IPv6. Exemplo: (www.exemplo.com, 2001:db8:10::1, AAAA).
- **MX**: Indica o servidor de correio de um domínio. Exemplo: (exemplo.com, mail.exemplo.com, MX).
- **CNAME**: Define um nome canônico para um apelido. Exemplo: (www.exemplo.com, servidor.exemplo.com, CNAME).
- **NS**: Aponta para o servidor DNS autoritativo de um domínio. Exemplo: (exemplo.com, dns.exemplo.com, NS).
- **PTR**: Usado em consultas inversas, mapeia um endereço IP a um nome de domínio. Exemplo: (192.0.2.1, www.exemplo.com, PTR).
- **SOA**: Define informações sobre a autoridade de um domínio, como o servidor primário e dados de delegação.
- **TXT**: Armazena texto arbitrário, usado para diversas finalidades, como verificação de domínio.

9.5. Funcionamento do cache de DNS

O cache DNS melhora o desempenho ao armazenar respostas de consultas por um período definido pelo TTL. Quando um servidor DNS recebe uma resposta, como o mapeamento de www.exemplo.com para um endereço IP, ele a armazena localmente. Se outra consulta para o mesmo nome chegar antes do TTL expirar, o servidor retorna a resposta em cache, evitando consultas adicionais à hierarquia DNS. Por exemplo, um servidor local pode armazenar endereços de servidores TLD, reduzindo o tráfego para servidores raiz. Após o TTL (geralmente dois dias), o registro é descartado, garantindo que alterações nos mapeamentos sejam refletidas eventualmente.

10. Protocolo HTTP e web

O *Hypertext Transfer Protocol* (HTTP) foi criado para permitir que clientes e servidores troquem informações de maneira prática e eficiente.

10.1. Clientes e servidores web

O HTTP é um protocolo baseado em trocas de mensagens de requisições e respostas entre clientes e servidores. No modo direto de acesso (sem proxy, por exemplo) o agente do cliente abre uma conexão TCP com o servidor web e através dessa conexão envia mensagens de texto (ASCII) com requisições. O cliente também envia ao servidor informações de negociação da transferência, como o idioma preferido por ele, a codificação esperada, se aceita compactação dos dados, os tipos de arquivo aceitos e se a conexão deve ser mantida ou encerrada após a transferência. Essas informações são acrescentadas na requisição, também em forma de texto, nas linhas de cabeçalho.

É importante lembrar que todas essas informações são enviadas por conexões TCP cujo modelo de comunicação é o de sequências de bytes, uma em cada direção.

De forma bem simplificada, o navegador web executa os seguintes passos para acessar uma página web estática:

1. Interpreta a URL fornecida pelo usuário
2. Identifica o nome do servidor contido na URL
3. Obtém o endereço IP do servidor através de uma consulta ao DNS
4. Abre uma conexão TCP com o servidor daquele endereço IP na porta 80 (padrão)
5. Usa o protocolo HTTP para enviar a solicitação de um arquivo HTML ao servidor
6. Recebe o arquivo HTML.
7. Interpreta o arquivo HTML e solicita aos respectivos servidores todos os objetos descritos em URLs contidas no arquivo HTML
8. Apresenta a página para o usuário

10.1.1. Conexões persistentes

Cliente e servidor podem manter as conexões TCP abertas para a solicitação e envio de vários objetos. Essas conexões são chamadas de conexões persistentes. Manter a conexão evita o atraso de abertura de novas conexões e pode superar a lentidão do controle de congestionamento do TCP (o TCP inicia a conexão transmitindo com uma taxa pequena e aumenta-a a medida que mais dados são enviados).

A técnica de *pipelining* permite ao cliente enviar múltiplas requisições numa mesma conexão TCP enquanto aguarda as respostas.

Menos conexões TCP significam uma menor latência e também menos uso de memória nos servidores e clientes e menos uso de CPU pelas partes. Porém, o problema de usar uma única conexão TCP para obter muitos objetos é que o atraso de quem está na frente da fila atrasará todos os envios subsequentes devido ao caráter sequencial da comunicação usando TCP.

A versão 2.0 do protocolo HTTP tenta resolver esse problema criando uma camada intermediária de comunicação que ao invés de utilizar o canal TCP diretamente para enviar os objetos, faz a multiplexação do canal quebrando as mensagens em pedaços pequenos e as enviando de forma intercalada. Assim, espera-se que aplicações web que utilizem a versão 2.0 do protocolo necessitem abrir um número menor de conexões TCP.

10.2. Formato das mensagens

As linhas das mensagens HTTP são separadas por uma sequência de dois caracteres especiais: *carriage return* (CR) e *line feed* (LF).

A resposta do servidor contém na primeira linha o status da resposta composto de um número e uma sentença legível que identificam o status. Exemplos de linhas de status são: “200 OK”, “404 Not Found”, “301 Moved Permanently” e “304 Not Modified”. Após a linha de status, seguem linhas contendo os cabeçalhos da resposta, seguidas de uma linha em branco e os bytes que compõe a entidade enviada, caso exista.

10.3. Cabeçalhos HTTP

Alguns tipos de cabeçalhos:

- Content-Length: tamanho do documento em octetos (bytes)
- Content-Type: tipo do documento
- Content-Encoding: codificação do documento
- Content-Language: idioma do documento

Como o HTTP suporta aquivos binários, não é possível a utilização de caracteres ou sequências especiais para delimitar o final dos objetos. Há duas maneiras de resolver o problema.

A primeira é usar o cabeçalho Content-Length para especificar o número de bytes do objeto. A segunda é usar um conteúdo chamado chunked.

No método de transferência chunked, o objeto sendo transferido pode ser dividido em partes (chunks). Cada parte começa com uma linha contendo unicamente um número em ASCII que representa o número de bytes da parte. Seguem então os bytes daquela parte. Um número zero como tamanho da parte indica o fim do objeto.

Outros exemplos de cabeçalhos:

- Connection: close

10.3.1. Exemplo: Negociação: do servidor, do navegador

```
Accept: text/html, text/plain; q=0.5, text/x-dvi; q=0.8
```

Requisições Condicionais:

```
If-Modified-Since: Mon, 01 Apr 2013 05:00:01 GMT
```

Esse cabeçalho é utilizado para o cliente informar o servidor qual é a data do objeto armazenado em seu cache. Caso o objeto não tenha sido modificado, ele não é devolvido pelo servidor e o cliente deve usar o objeto que está no cache.

10.4. Cookies

O protocolo HTTP não mantém o estado de conexões (*stateless*) e isso significa que cada requisição do cliente deve conter informações completas sobre o que ele está requisitando. Sites que precisam guardar informações sobre os usuários que os navegam, como sites onde os usuários fazer login, tão comuns hoje em dia, requerem a utilização de um recurso previsto no HTTP e chamado de cookies.

Cookies funcionam da seguinte maneira: quando o cliente acessa o site pela primeira vez, o servidor envia para ele um cabeçalho chamado set-cookie seguido de uma string que é o cookie. Quando o cliente faz uma nova requisição ao mesmo site, o cookie é enviado para o servidor em um cabeçalho de nome cookie. Dessa maneira, o cookie pode conter a identificação do usuário.

Cabe a aplicação web armazenar as informações de credencias dos usuários e todas as informações relativas ao contexto do acesso do usuário ao site, não sendo esse armazenamento responsabilidade do HTTP.

É importante que sites que utilizem cookies para identificar usuários usem conexões seguras para que a criptografia garanta que os cookies não sejam interceptados e usados por terceiros para se passarem pelo usuário do site. Essa utilização malicioso dos cookies é chamada de sequestro de sessão.

10.5. Proxies web

O HTTP descreve o suporte para cache de dados e para proxies web. Os proxies web são programas intermediários entre clientes e servidores cuja principal função é fazer cache de dados e reduzir o tráfego de dados no enlace de acesso do cliente.

Além do método GET, o cliente pode enviar solicitações utilizando um dos métodos a seguir: HEAD, POST, PUT, POST, DELETE e OPTIONS. O método HEAD é análogo ao

GET, porém o servidor devolve apenas o cabeçalho, sem devolver o objeto associado à requisição. O método POST permite o envio de dados para serem processados pelo servidor. O método PUT serve para fazer UPLOAD de uma representação para uma URI. O comando DELETE serve para remover um recurso do servidor. O método OPTIONS devolve uma lista com os métodos aceitos pelo servidor.

10.6. Versões

Versões 1, 1.1, 2, 3.

O HTTP foi inicialmente concebido por Tim Berners-Lee em 1989 no CERN, junto com o HTML, com o objetivo de transferir arquivos, buscar índices de documentos e permitir negociações entre clientes e servidores. Antes do HTTP, protocolos como FTP (1971) e SMTP (1981) eram usados para transferência de arquivos e e-mails na ARPANET e na Internet. A primeira implementação, HTTP 0.9 (1990-1991), era simples, suportando apenas requisições GET para documentos HTML, com respostas em ASCII e conexões fechadas após cada transferência.

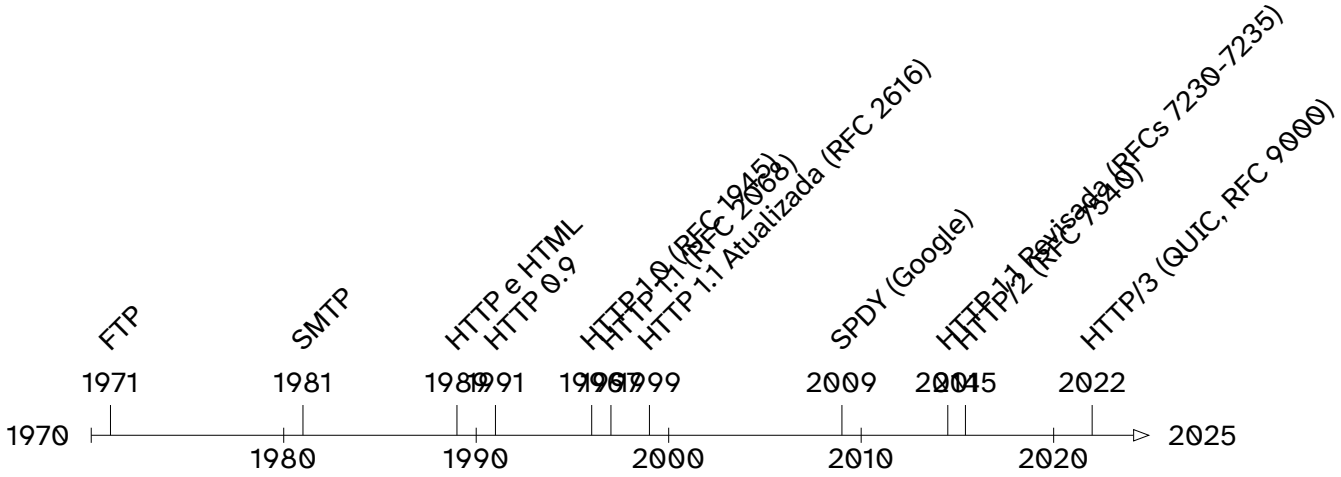
Com o crescimento da Web, o protocolo evoluiu:

- HTTP 1.0 (1996): Formalizado na RFC 1945, introduziu métodos como POST e HEAD, cabeçalhos para metadados, negociação de tipo de mídia e suporte a proxies, mas usava conexões não persistentes por padrão.
- HTTP 1.1 (1997-1999): Definido na RFC 2068 e atualizado na RFC 2616, trouxe conexões persistentes, pipelining, autenticação, transferência chunked e caching avançado.

Em 2014, foi revisado e dividido em seis RFCs (7230 a 7235), refinando aspectos como sintaxe, semântica e caching.

- HTTP/2 (2015): Baseado no SPDY (2009) do Google e formalizado na RFC 7540, focou em eficiência com multiplexação, compressão de cabeçalhos, push do servidor e enquadramento binário, reduzindo a latência e o uso de conexões TCP.
- HTTP/3 (2022): Utiliza o QUIC sobre UDP (RFC 9000, 2021), oferecendo multiplexação sem bloqueio de linha, criptografia obrigatória, conexões rápidas (0-RTT/1-RTT) e suporte a mobilidade, resolvendo limitações do TCP como o bloqueio head-of-line.

A Web, inicialmente limitada a texto e imagens, passou a suportar multimídia, scripts e páginas dinâmicas, transformando a Internet em uma rede essencial para comunicação e comércio.



10.6.1. Tecnologias do HTTP e Sua Evolução

As tecnologias incorporadas ao HTTP ao longo de suas versões refletem a necessidade de maior eficiência, segurança e suporte a aplicações web modernas.

- **Métodos HTTP**: Métodos como GET, POST e HEAD (HTTP 1.0) permitiam operações básicas de recuperação e envio de dados. O HTTP 1.1 expandiu com PUT, DELETE, OPTIONS, TRACE e CONNECT, habilitando aplicações RESTful e túneis seguros (HTTPS).
- **Conexões Persistentes**: Introduzidas no HTTP 1.1 com o cabeçalho `Connection: keep-alive`, reduziram a sobrecarga de abrir/fechar conexões TCP. O HTTP/2 aprimorou isso com multiplexação, e o HTTP/3, via QUIC, adicionou reconexão rápida (0-RTT/1-RTT) e suporte a redes móveis, minimizando latência em cenários dinâmicos.
- **Pipelining**: Disponível no HTTP 1.1, permitia enviar múltiplas requisições sem esperar respostas, mas sofria com bloqueio head-of-line (HOL). O HTTP/2 substituiu o pipelining por multiplexação, e o HTTP/3 eliminou o HOL completamente com QUIC, garantindo processamento paralelo eficiente.
- **Multiplexação**: Introduzida no HTTP/2, permite múltiplos fluxos bidirecionais em uma única conexão TCP, usando enquadramento binário. O HTTP/3 aprimorou com QUIC sobre UDP, eliminando HOL no nível de transporte e aumentando a resiliência em redes instáveis.
- **Compressão de Cabeçalhos**: O HTTP/2 implementou o HPACK, que usa tabelas dinâmicas e codificação Huffman para reduzir o tamanho dos cabeçalhos, diminuindo latência. O HTTP/3 usa QPACK, adaptado para QUIC, mantendo eficiência e adicionando resiliência a perdas de pacotes.
- **Push do Servidor**: Introduzido no HTTP/2, permite que servidores enviem recursos (ex.: CSS, JS) proativamente, reduzindo latência. O HTTP/3 mantém essa funcionalidade, otimizada com QUIC para entrega rápida, especialmente em páginas web complexas.
- **Protocolo de Transporte**: O HTTP 1.0 e 1.1 usam TCP, que, embora confiável, sofre com latência em conexões iniciais e HOL. O HTTP/2 otimiza o TCP com enquadramento binário, enquanto o HTTP/3 adota QUIC sobre UDP, integrando criptografia, multiplexação e conexões rápidas, ideal para redes modernas.
- **Caching**: O HTTP 1.0 oferecia caching básico com `Last-Modified`. O HTTP 1.1 introduziu `ETag`, `Cache-Control` e `Expires`, otimizando reuso de recursos. O HTTP/2 e HTTP/3 mantêm esses mecanismos, com push do servidor e QUIC reduzindo a necessidade de requisições repetitivas.
- **Criptografia**: Inicialmente opcional (SSL no HTTP 1.0, TLS no 1.1), tornou-se recomendada no HTTP/2 com TLS 1.2 e obrigatória no HTTP/3 com QUIC e TLS 1.3.
- **Transferência Chunked**: Introduzida no HTTP 1.1, permite streaming de dados em blocos, ideal para respostas dinâmicas. O HTTP/2 e HTTP/3 mantêm essa funcionalidade, com enquadramento binário e QUIC.

Feature	HTTP 1.0	HTTP 1.1	HTTP/2	HTTP/3
Métodos	GET, POST e HEAD	PUT, DELETE, OPTIONS, TRACE e CONNECT. Suporta APIs RESTful.	Mesmos métodos do 1.1	Mesmos métodos do 2.0
Conexões Persistentes	Não suportado. Cada requisição abre e fecha uma conexão TCP	Introduz `Connection: keep-alive`, permitindo múltiplas requisições na mesma conexão TCP	Melhora persistência com multiplexação	Persiste conexões via QUIC, com reconexão rápida (0-RTT/1-RTT)
Pipelining	Não suportado	Permite enviar múltiplas requisições sem esperar respostas, mas sofre com bloqueio head-of-line (HOL), atrasando objetos subsequentes.	Substituído por multiplexação, que elimina HOL ao processar requisições/respostas em paralelo via fluxos independentes.	Não usa pipelining. Multiplexação via QUIC elimina HOL completamente
Multiplexação	Não suportado. Cada requisição é processada sequencialmente	Não suportado. Pipelining tenta mitigar, mas HOL persiste	Sim, via fluxos bidirecionais em uma única conexão TCP, permitindo processamento paralelo de req/res. Usa enquadramento binário.	Multiplexação aprimorada com QUIC sobre UDP, eliminando HOL no nível de transporte e suportando fluxos independentes

Feature	HTTP 1.0	HTTP 1.1	HTTP/2	HTTP/3
Compressão de Cabeçalhos	Não suportado. Cabeçalhos em texto puro (ASCII)	Não suportado. Cabeçalhos continuam em texto puro	Usa HPACK, com tabelas dinâmicas e codificação Huffman	Usa QPACK, mantendo compressão com tabelas dinâmicas e codificação Huffman
Push do Servidor	Não suportado. Clientes devem solicitar todos os recursos explicitamente	Não suportado. Mesma limitação do 1.0	Permite que o servidor envie recursos proativamente antes da solicitação do cliente	Mantém push do servidor, mas otimizado com QUIC
Protocolo de Transporte	Usa TCP na porta 80, com conexão estabelecida para cada req	Usa TCP, com conexões persistentes na porta 80 ou 443 (HTTPS)	Usa TCP (com TLS na 443), com enquadramento binário e multiplexação	Usa QUIC sobre UDP (443), com criptografia integrada, multiplexação sem HOL e suporte a conexões rápidas (0-RTT).
Caching	Suporte básico com `Last-Modified` e `If-Modified-Since`	Avançado com `ETag`, `Cache-Control`, `Expires` e validação condicional	Mesmos mecanismos do 1.1, mas otimizados com push do servidor e multiplexação	Mesmos mecanismos, mas com QUIC
Criptografia	Opcional, via SSL inicial (HTTPS na porta 443), mas não amplamente utilizado	Opcional, com TLS substituindo SSL. HTTPS ganha adoção, mas muitos sites	Recomenda TLS 1.2 ou superior, com HTTPS como padrão na maioria	Criptografia obrigatória via QUIC, integrada ao protocolo, com TLS 1.3

Feature	HTTP 1.0	HTTP 1.1	HTTP/2	HTTP/3
		ainda usam HTTP puro.	dos navegadores	
Transferência Chunked	Não suportado. Conteúdo é enviado com tamanho fixo via `Content-Length`	Introduz transferência chunked, permitindo envio de dados em blocos de tamanho variável	Mantém chunked, otimizado com enquadramento binário	Mantém chunked com QUIC

11. Neutralidade de rede

Conceito de neutralidade da rede Neutralidade da rede no marco civil da Internet.