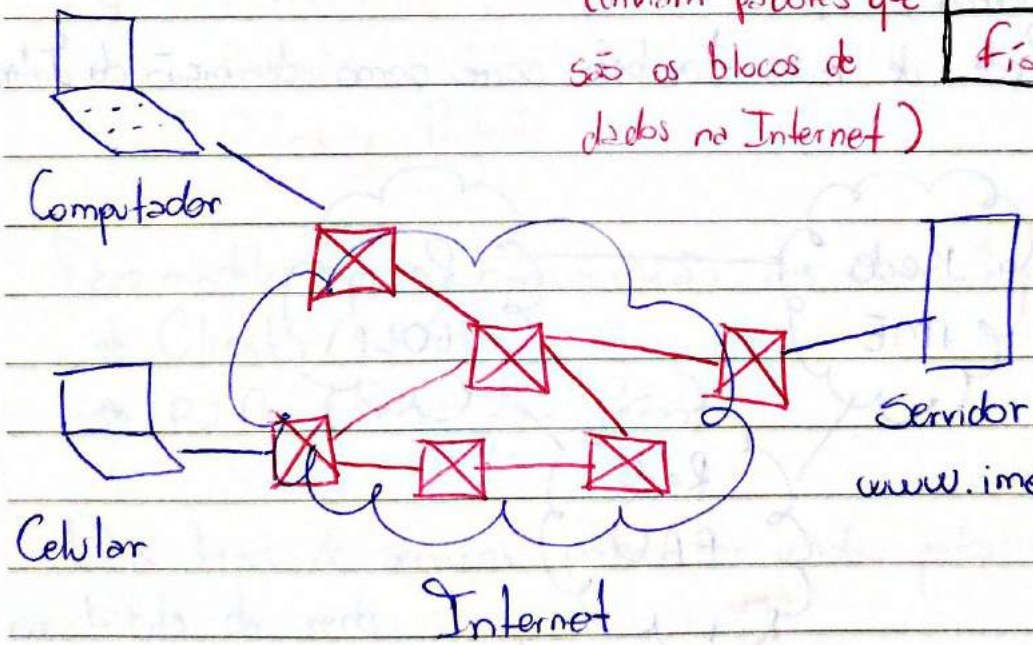
01/08

- O que é Internet?

Arquitetura TCP/IP

- ~~Milhões~~ Bilhões de elementos de computação (Internet) interligados.
  - ↳ Hospedeiro ou Host ou Nó ou sistema final (Executam as Aplicações)
  - ↳ Roteadores (enviam pacotes que são os blocos de dados na Internet)

Aplicação
Transporte
Rede
Enlace
Física



⊠ → Roteadores

□ → Hosts

- Aplicações Distribuídas
- Enlaces de Comunicação: fibra óptica / fio de cobre / rádio / satélite
  - ↳ Links

/ /

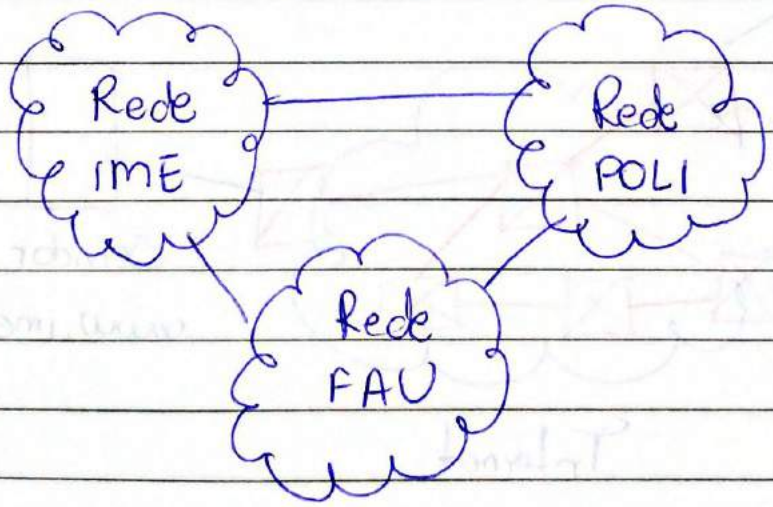
(largura de banda)
↳ capacidade de transmissão em bps

- Protocolos definem como as aplicações distribuídas devem se comunicar.

Aplicação: HTTP / FTP
Transporte: TCP
Rede: IP

} Definições em RFC (Request for Comments) Organizados pela IETF

- Rede de redes também serve como definição da Internet



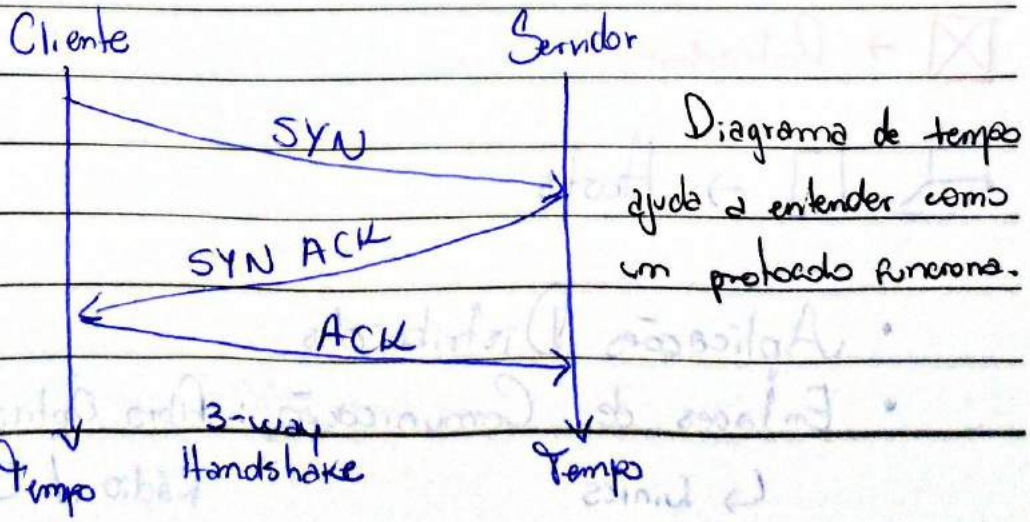
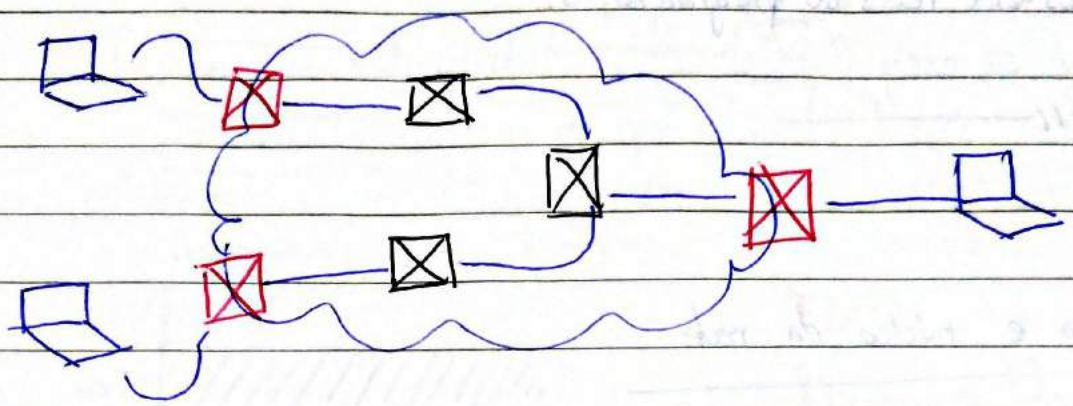


Diagrama de tempo ajuda a entender como um protocolo funciona.

- Borda e Núcleo da Rede



⊠ → Roteador de Borda
Roteador Padrão
Gateway Padrão

⊠ → Roteador de Núcleo

Dois modelos para comunicação dos elementos na borda:
* Cliente / Servidor
* P2P (Peer-to-Peer)

Dois tipos de serviços podem ser usados pelos elementos na borda da rede:
* Orientado à conexão: TCP (RFC 793) → HTTP FTP
  - Transferência de dados confiável e sequencial
  - Controle de fluxo
  - Controle de Congestionamento
* Não orientado à conexão: UDP (RFC 768) → DNS streaming de vídeo
  - "Melhor Esforço"
  - Costuma ser mais rápido que o TCP principalmente para poucos dados.

/ /

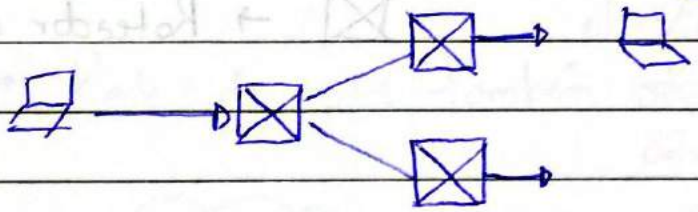
O sniffer de redes é um "depurador" para quem programa protocolos em redes

— // —

08/03

Borda e núcleo da rede

Núcleo da rede é uma malha de roteadores interconectados.



Duas formas de definir como a rede será usada por todas as comunicações.

- Comutação de circuitos → um canal dedicado para cada conexão.
- Comutação de pacotes → dados são enviados em blocos discretos. (Internet é assim) ☺

Comutação de Circuito

→ Largura de banda, principalmente

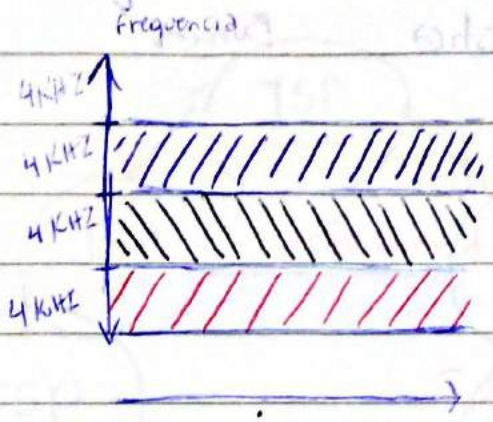
Recursos são alocados para cada comunicação e são reservados fim-a-fim no início da comunicação

→ Depende de protocolos de sinalização e de roteamento

Comutação de Pacotes

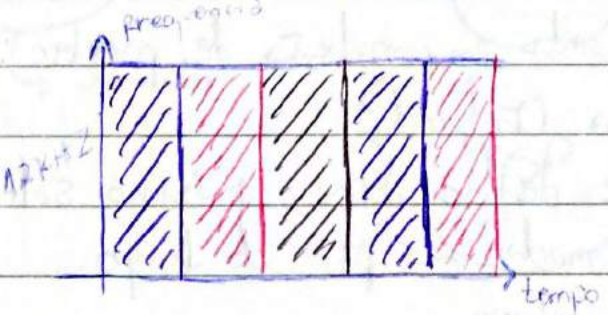
Recursos sempre compartilhados

Recursos são divididos em "pedaços" para as várias comunicações sem garantia de que a qualidade requisitada será entregue.



FDM
Frequency Division Multiplexation

Comutação de Circuito

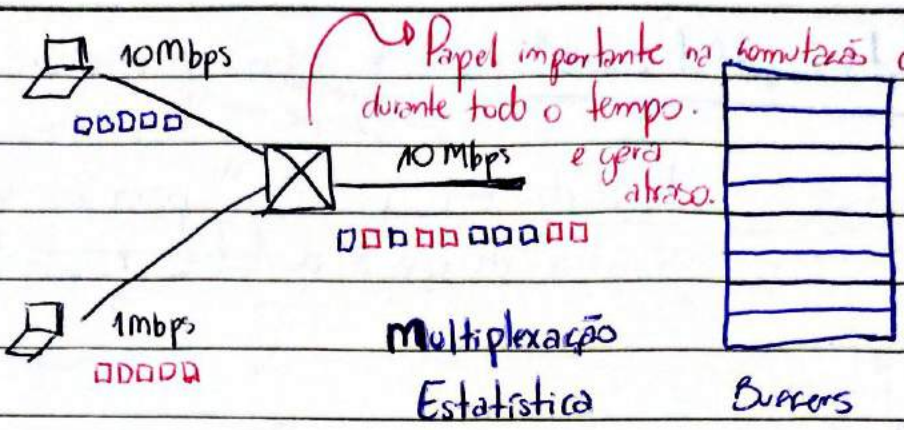


TDM
Time Division Multiplexing

Comunicações são divididas em pedaços menores chamados de pacotes.

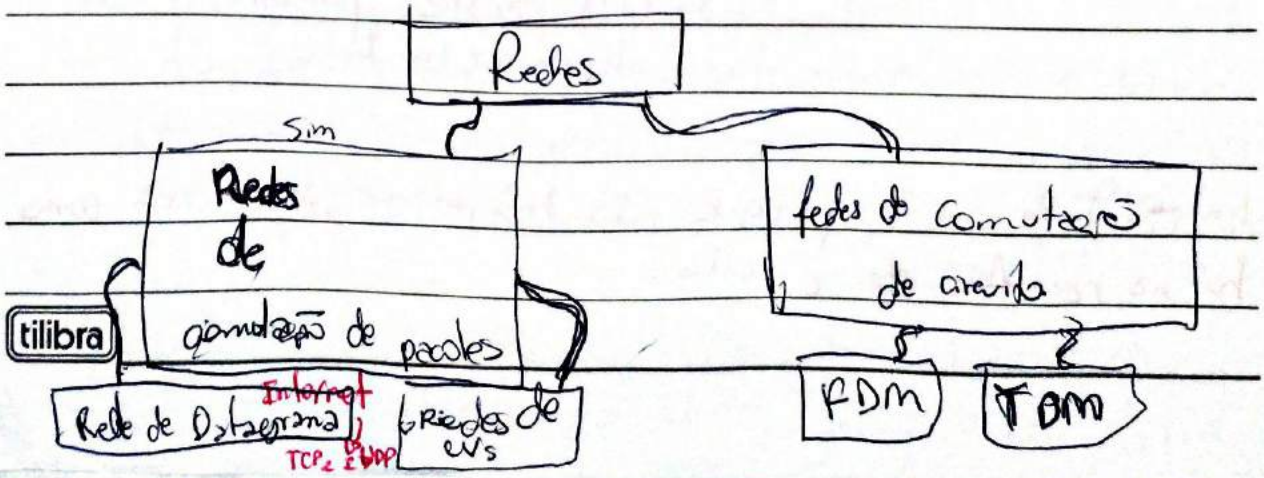
↳ cada pacote percorre a rede até chegar ao destino.

→ Tem colisão porque não há reserva de recursos como há na comutação de circuitos.



Atraso → Processamento Nodal
Fila esperando pela transmissão
Transmissão
Tempo de propagação

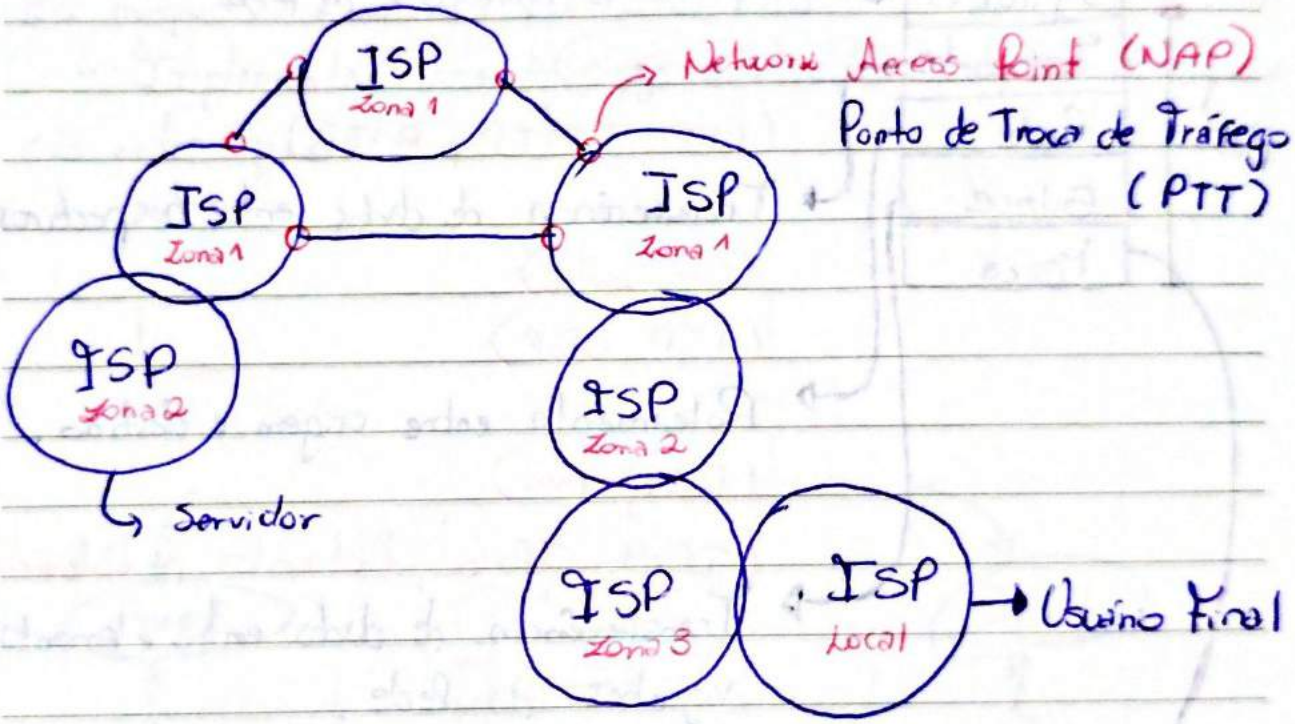
Dois tipos de redes baseadas em comutação de pacotes:
- Rede de Datagrama (Internet)
  - O endereço de destino define o próximo salto.
  - Rotas podem mudar ao passar do tempo.
- Rede de circuitos virtuais
  - Cada pacote leva um número (ID do circuito virtual) e esse número define o próximo salto.
  - O caminho é FIXO!



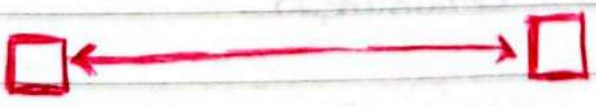
13/03

Estrutura da Internet

- Grosseiramente Hierárquica



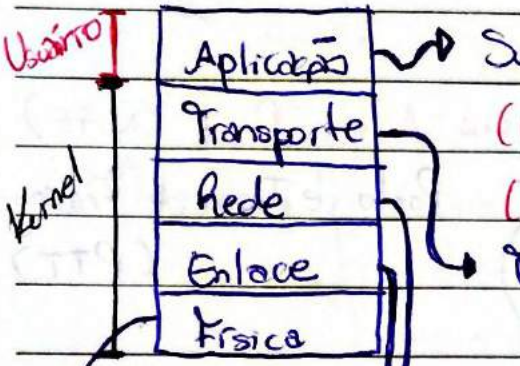
Protocolos de roteamento no núcleo da rede não são os mesmos na borda da rede.



Arquitetura em Camadas

/ /

- Organizar melhor o modo de resolver cada problema necessário para comunicação entre 2 processos.
- Uma boa solução para resolver problemas complexos e com entidades heterogêneas.



Aplicação → Suporta as aplicações de rede.
(Mensagens)
(FTP, SMTP, HTTP)

Transporte → Transferência de dados entre hospedeiros.
(Segmentos)
(TCP, UDP)

Rede → Roteamento entre origem e destino.
(Datagramas)
(IPv4, IPv6, Protocolos de Roteamento)

Enlace → Transferência de dados entre elementos vizinhos da Rede.
(Quadros)
(Wi-Fi, Ethernet)

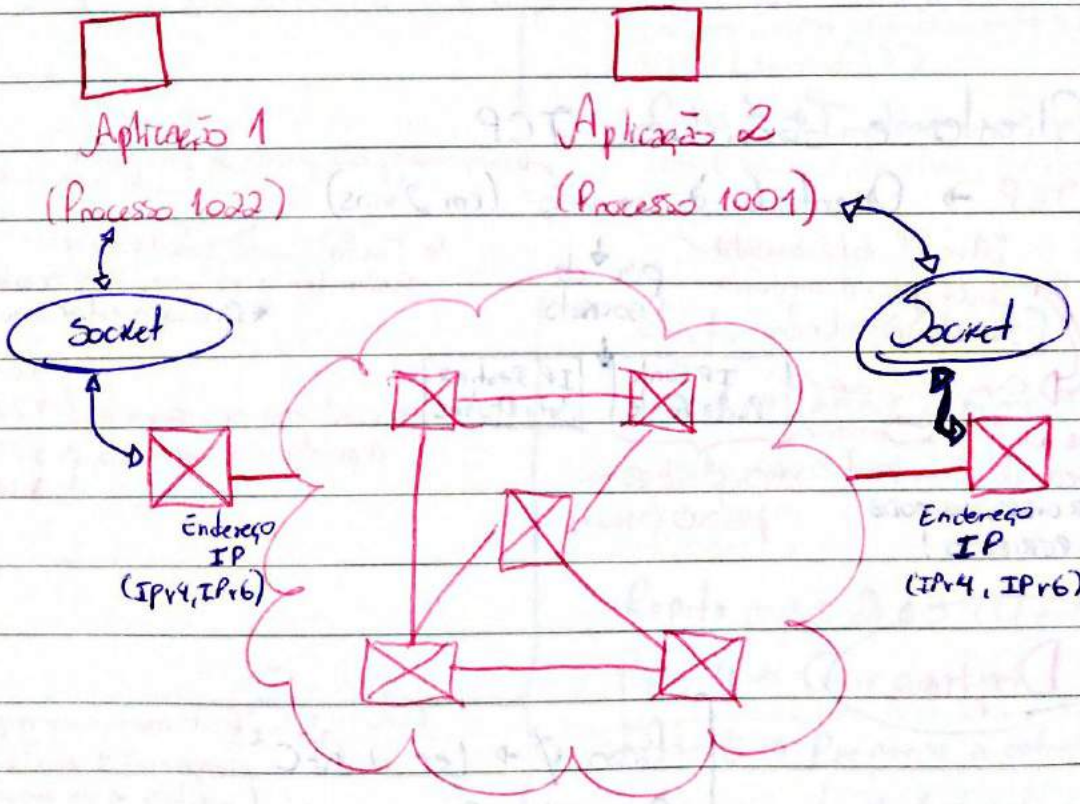
Física → Transmite os bits nos enlaces de comunicação.

Ossificação na Internet ⇒ Dificuldade em mexer no núcleo da rede.

15/03

Camada de Aplicação

- Aplicações são processos locais dos sistemas operacionais das máquinas envolvidas na comunicação.
- Implementar uma aplicação em rede é realizar comunicação entre processos.



A identificação do processo é feita por portas:

65536 portas TCP → Ex.: 80 HTTP, 22 SSH, 25 SMTP

65536 portas UDP → Ex.: 53 DNS

/ /

27/03

- Softwares para "debug" de códigos de redes

NMAP (IP LocalHost)
- Retorna portas abertas na máquina

NETSTAT -anp | less
- Processos com conexões abertas na máquina

LSOF -n | less (| grep -i IPv)
- Lista arquivos abertos na máquina
(Trata tudo como arquivos, inclusive sockets)

- Clientes e servidores TCP

TCP → Orientado à conexão (em 2 vias) → Programador não precisa se preocupar se o pacote não chegar

"A rede" está enviando dados para a memória do computador!

O mínimo que deve ser feito é limitar a área de memória! BUFFER OVERFLOW PODE SER PERIGOSO!

par de sockets

IP Fonte	IP Destino
Porta Fonte	Porta Destino

Porta Fonte: definida de forma pelo S.O (>1024)

Porta Destino: costuma ser fixa e dependente da aplicação
- 80 HTTP
- 22 SSH
- 25 SMTP
- 110 POP3

Tanto cliente quanto servidor podem ler e escrever nos sockets (read/write)
* Precisam estar sincronizados

Daytime

Cliente

Passo 0 → Ler a RFC (Porta recomendada é a 13 TCP)

Passo 1 → Definir quantos (1 socket) int sockfd sockets serão necessários
(Sequência de read's ainda write's → vai ter um read no código)
(Quem fecha a conexão → para fazer o close corretamente)

Passo 2 → Definir as áreas de memória (1 Buffer)

necessárias para armazenar informação char recvline [MAXLINE+1]

/ /

Cliente

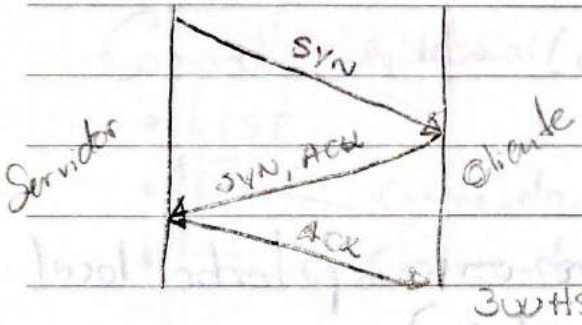
Passo 3 → Declarar uma estrutura de endereçamento para cada socket
struct sockaddr_in servaddr (1 Estrutura)

Passo 4 → Criar cada socket
sockfd = socket (AF_INET, SOCK_STREAM, 0)

Passo 5 → Preparar a estrutura de endereçamento
bzero (&servaddr, sizeof (servaddr))
servaddr.sin_family = AF_INET
servaddr.sin_port = htons(13).
inet_pton (AF_INET, argv[1], servaddr.sin_addr)

Passo 6 → Conectar no servidor (Fazer o 3WHS)
connect (socket, (struct sockaddr *)&servaddr, sizeof (servaddr))

Passo 7 → Implementar a aplicação
read (sockfd, recvline, MAXLINE)



Servidor

Repita passo 0 (cliente)

Passo 1 → Definir o número de sockets necessários. (Pelo menos 2, 1 que vai ficar escutando e 1 que vai ser definido em tempo de execução)
int listenfd;
int connfd;

Repita passos 2 e 3 (Cliente)

Passo 4 → Criar o socket
listenfd = socket(AF_INET, SOCK_STREAM, 0)

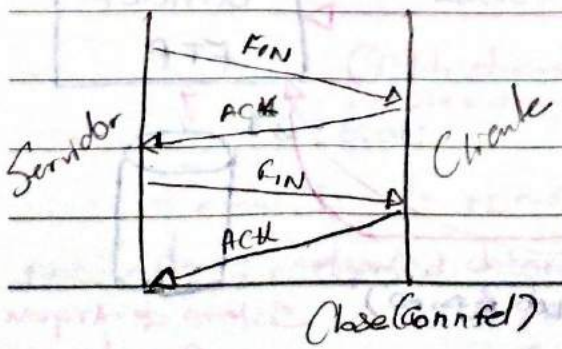
Passo 5 → Preparar a estrutura de endereçamento do lado do servidor
servaddr.sin_addr.s_addr = htonl (INADDR_ANY)

Passo 6 → Porta bem definida
bind(listenfd, (struct sockaddr*)servaddr, sizeof(servaddr))

Passo 7 → Esperar por conexões
listen(listenfd, LISTENQ)

Passo 8 → Aceita conexão de um cliente
connfd = accept(listenfd, (struct sockaddr*)NULL, NULL)

Passo 9 → Aplicação
write(connfd, buff, strlen(buff)) / close(connfd)

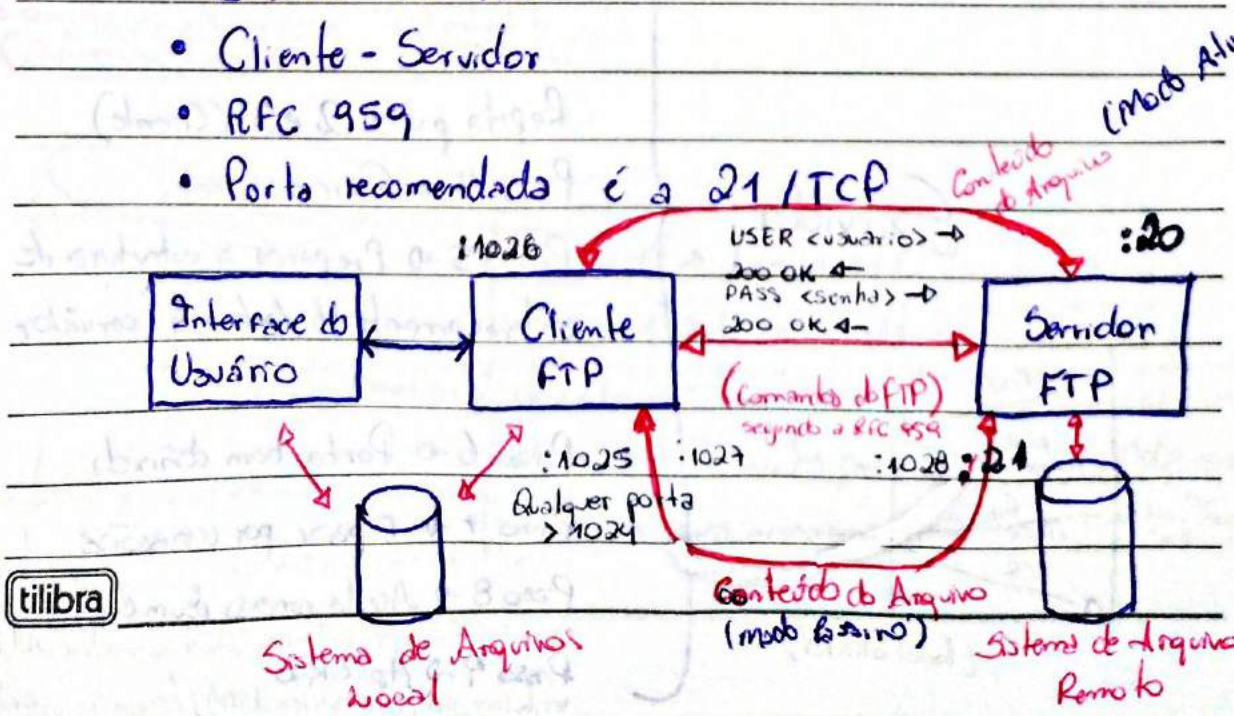


/ /

29/03

- Outros clientes com sockets HTTP/FTP/DNS
  - HTTP → HEAD foi usado para pegar informações básicas do servidor - Host local - Cookie - Nome do Servidor
  - FTP → USER e PASS são dois comandos que enviam usuário e senha.
  - DNS → É um protocolo de aplicação em cima do UDP
    - Não uso sockets explicitamente porque uso a função gethostbyname
    - Cache é mantido para não inundar a rede.

- FTP (File Transfer Protocol)
  - Transferência de arquivos entre um computador local e um computador remoto (nos dois sentidos)
  - Baseado no TCP
  - Cliente - Servidor
  - RFC 959
  - Porta recomendada é a 21/TCP



/ /

Na porta 21 → Canal de controle

Dois modos para transferir arquivos → Canal de Dados

- Modo ativo: Servidor vindo da porta 20 estabelece uma nova conexão com o cliente para enviar ou receber o arquivo.
- Modo passivo: Cliente especifica uma porta alta e avisa essa porta para o servidor que conecta também vindo de uma porta alta.

2 Sockets → 1 para o canal de controle
2 Sockets → 1 para o canal de dados

Comandos enviados (Cliente → Servidor)

- LIST
- RETR <nome-do-arquivo>
- STOR <nome-do-arquivo>

- DNS (Domain Name System)

Endereço IP
(IPv4 p/ex. 32b)
(IPv6 p/ex. 128b)

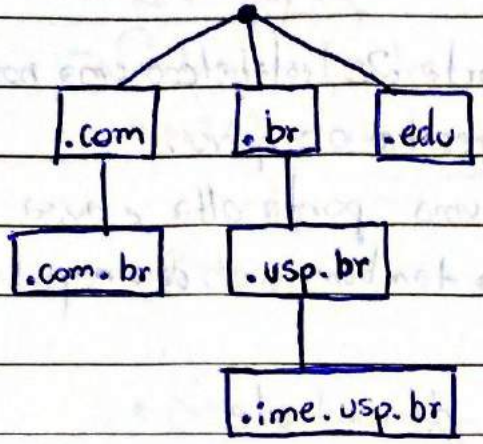
www.google.com ⇄ DNS ⇄ 200.xx.yy.33

Antigamente → Arquivo texto mapeando nome → IP
/etc/hosts (Ainda existe! Serve p/ resolver nomes localmente)

Resolução de nomes é algo que é feito o tempo todo seguindo a ordem: - cache local da máquina
- /etc/hosts
- Servidor DNS → Definido em /etc/resolv.conf.

/ /

Vários servidores raiz — nslookup, dig } Resolução de nomes



whois (infos sobre domínio)

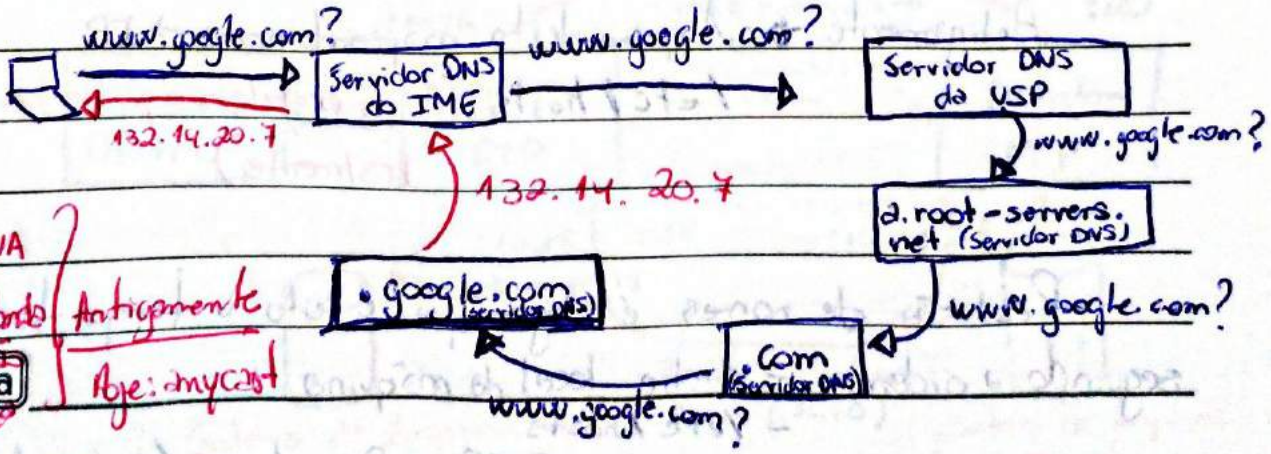
03/04

Servidor DNS → BIND

- DNS — RESOLVER = Cliente DNS → Embutido no

Ao fazer uma requisição: SO por meio de chamadas de sistema e bibliotecas.

- o computador acessa um servidor DNS (/etc/resolv.conf)
  Pacote UDP na porta 53 do servidor
- esse servidor DNS muitas vezes é responsável por um domínio inteiro.
- esse servidor DNS também conhece outros servidores, em particular, os outros 13 root domain servers
  http://root-servers.net ↳ a.root-servers.net ... m.root-servers.net



1º EVA
1 Aobrdo
Antigamente
Hoje: anycast
tilibra

/ /

Mensagem DNS carrega perguntas sobre domínios (cliente envia essas mensagens) Existem também resoluções reversas IP → Domínio (nome)

e respostas com endereços IP associados a um nome.

(servidores enviam essas mensagens)

16 bits	16 bits
Identificação	Flags
Número de Perguntas	Número de Respostas
...	...
Perguntas	
Respostas	
...	

Mensagem DNS

Respostas possíveis: A → Address

MX → Mail Exchange

(Servidor de email responsável pelo domínio)

- HTTP

http://bla.com:8080/prog.html

8080 TCP

http://www.usp.br/ → vazio, espera resposta padrão do servidor (ex. index.php / index.html)

URL → Base do HTTP

↳ Qual a máquina?

↳ Qual a porta a ser acessada?

↳ Qual o recurso solicitado?

8080 TCP

prog.html

tilibra

/ /

HTTP. 1.0 → RFC 1945 (não persistente)
HTTP 1.1 → RFC 2068 (persistente)

- TCP (Por padrão na porta 80)
- Cliente - Servidor

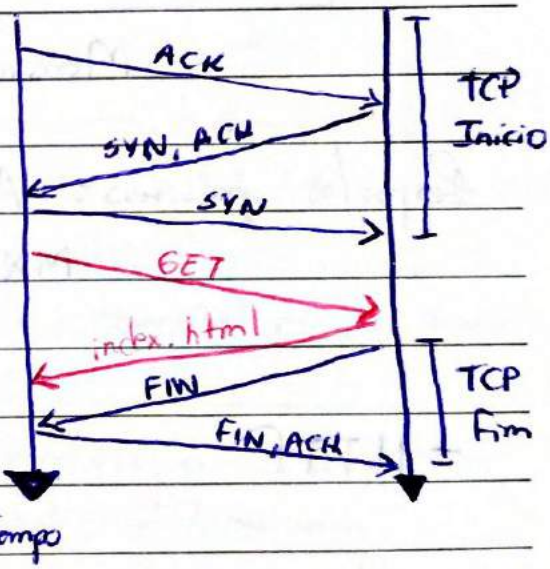
↙ Navegador Web
Chrome, Firefox → Protocolo sem estado

↘ Apache
NginX

↳ Por causa disso, é necessário em muitos casos implementar mecanismos de rastreio como COOKIES.

http://www.ime.usp.br/~batista
↓
index.html

HTTP 1.0 → Cada objeto requer 1 ~~conexão~~ requisição por conexão TCP



HTTP 1.1 → 1 requisição para todos os objetos da mesma máquina.

tilibra (com pipeline) → Possível fazer requisições sem receber respostas

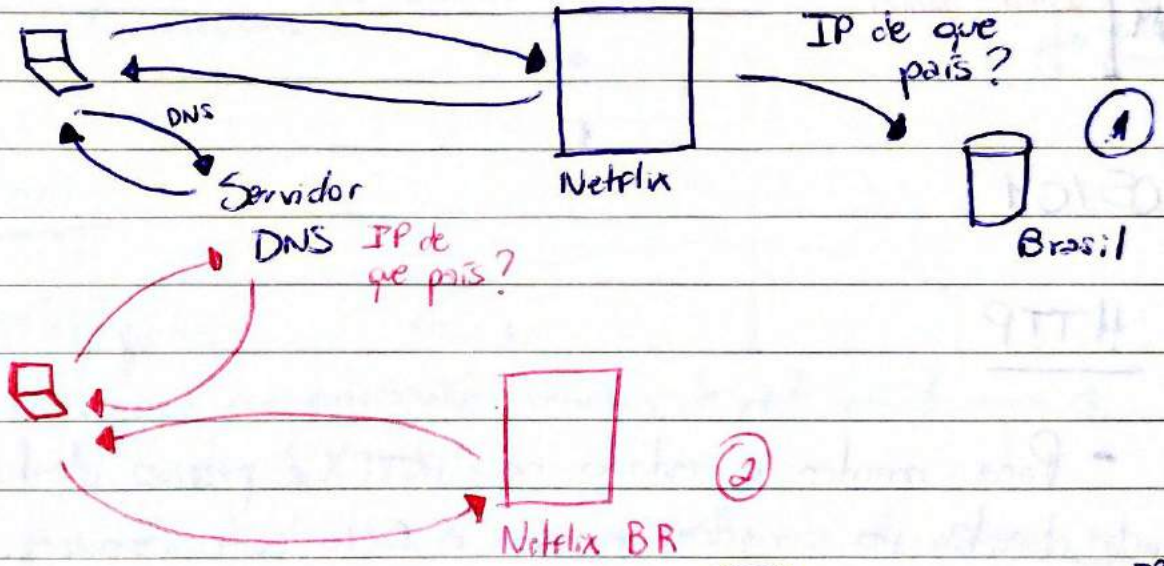
GET

/ /

Redes (CDN) Ex.: akamai.net

Para identificar a origem:

1) Identifica o IP do Cliente (Servidor HTTP)
2) Identifica o IP do cliente e devolve pro cliente o servidor com o conteúdo correto (Servidor DNS)



Mensagem HTTP
- são em ASCII (Por isso dá para reproduzir usando TELNET no terminal)
- Requisição: GET, POST, HEAD, DELETE, ...
  - GET <recurso> HTTP/1.0\n
    User-agent: Mozilla/4.0\n
    Accept: text/html, image/jpg\n
    Accept-language: Fr\n
    \n
  - pra 1.1
- Resposta
  - 301 Redirecionamento
  - 404 Recurso Inexistente
  - HTTP/1.0 200 OK
    Date: Thu, 06 Aug 1998 12:00:15 GMT
    Server: Apache/1.3.0 (Linux)
    Last-Modified: Mon, 22 jun 1998 ← se isso passou ou não usa o cache
    Content-Lenght: 6821
    Content-Type: text/html
    ::
    Dados

GET → Variáveis que definem o conteúdo tem os valores atribuídos na URL.

http://ime.usp.br/index.html?nome=daniel&senha=daniel

/ /

POST → Para ocultar a informação da URL

POST index.html HTTP/1.1 \n

⋮

Entity Body
name = daniel
senha = daniel

05/04

HTTP

- Para manter o estado no HTTP, é preciso identificar cada cliente no servidor e isso é feito com cookies.

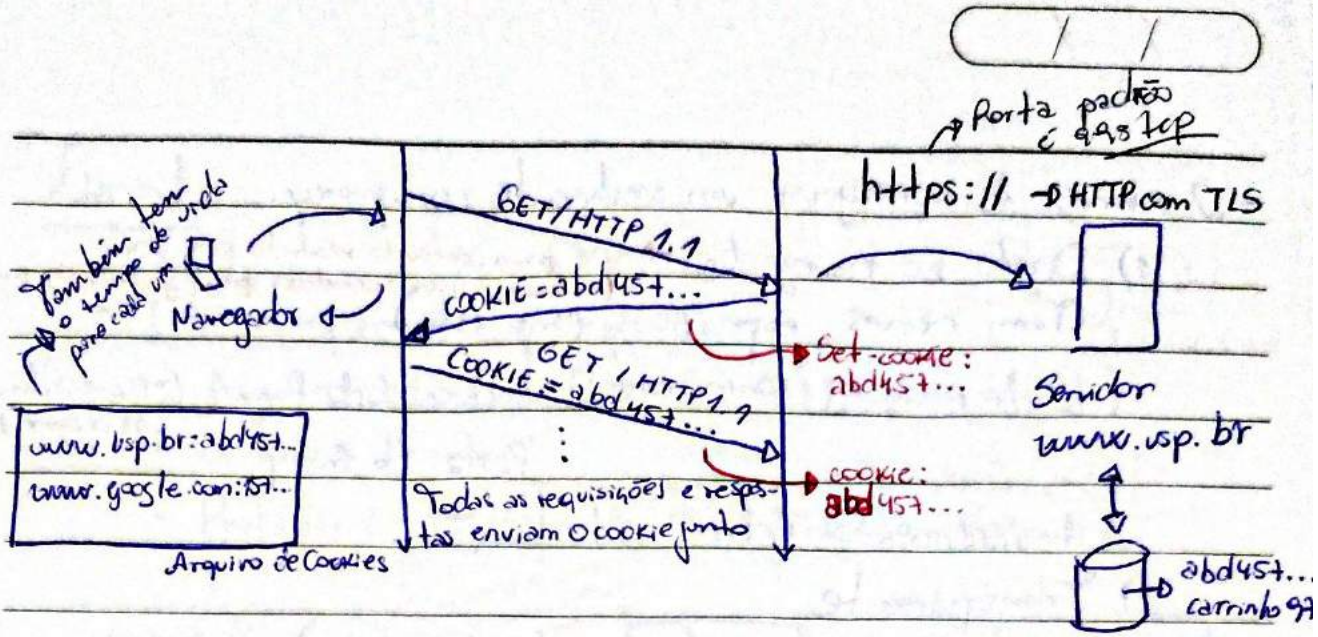
COOKIES
- Linha de cabeçalho do cookie na mensagem HTTP response
- Linha de cabeçalho de cookie na mensagem HTTP request
- Arquivo de cookie mantido no navegador e manipulado ~~rastreado~~ a cada requisição pro servidor
- Banco de dados do lado do servidor mantendo os cookies e o estado.

Observações:

1) Privacidade.

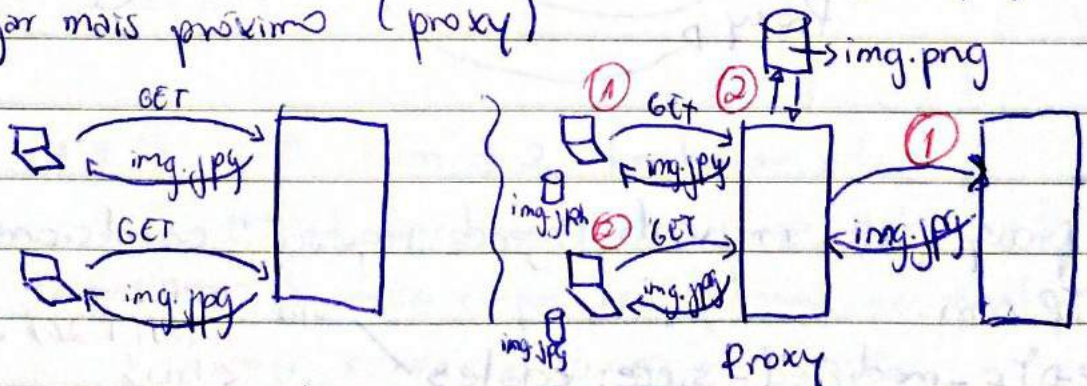
tilibra 2) Segurança do ponto de vista de acesso não autorizado.

3) Dificulta um pouco scripts baseados em páginas web. {wget curl



Proxy

- Web cache
- Útil para economizar os recursos de rede em termos de largura de banda (X Mb/s)
- Ao invés de baixar o mesmo conteúdo da origem n vezes, baixa 1 vez e nas outras n-1 vezes, pega de um lugar mais próximo (proxy)



Exemplo de servidor proxy é SQUID ← PROXY HTTP → Geralmente o proxy HTTP funciona na porta 3128 TCP

Pontos Positivos:

- Pro servidor: economia de recursos pois serve 1 cliente (Proxy) ao invés de n.
- Pros clientes: economia de tempo porque baixa o conteúdo de um ponto mais próximo.

/ /

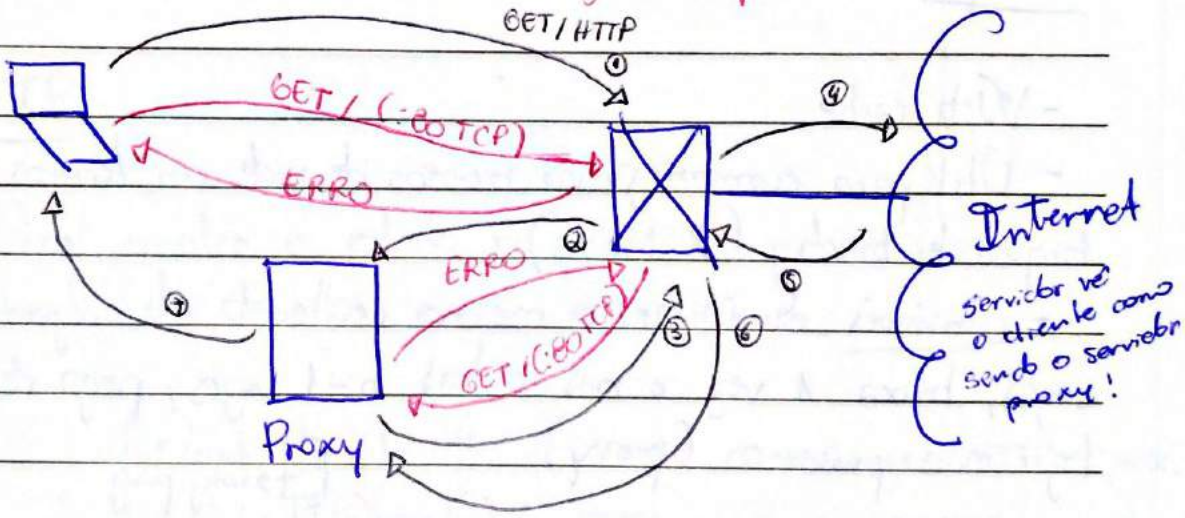
- 2 formas de configurar um ambiente com proxy.

1) Direto no navegador → Útil para conteúdo restrito por país (Existem vários servidores proxy gratuitos por país)
(Tem menus específicos para isso)

2 informações no mínimo: Endereço do Proxy (IP que acham vai dizer)
Porta do Proxy

1 informação extra: autenticação

2) Transparente
(Depende de um redirecionador de pacotes)
↳ Geralmente vai ser o gateway padrão



- O proxy pode ser usado fazendo requisições condicionais no HTTP com:

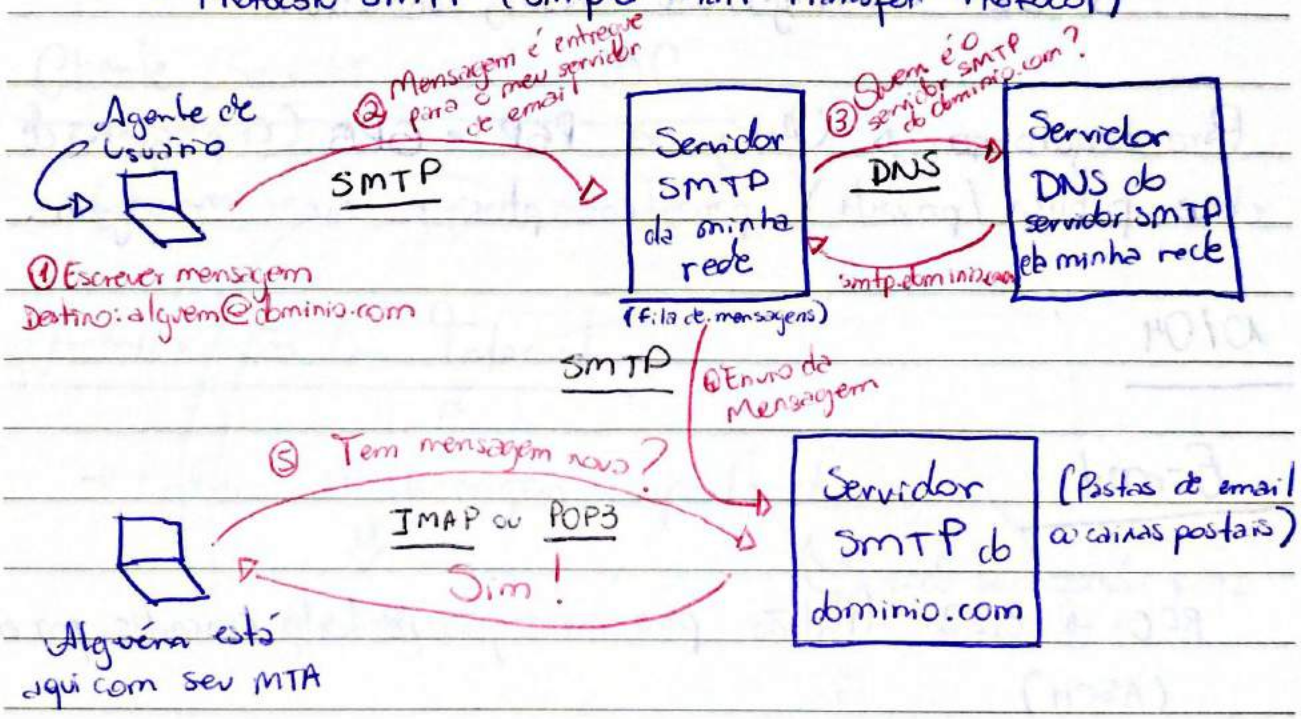
Cliente → If-modified-since: <date> (Servidor Proxy)

- sim → HTTP Response
HTTP/1.1 200 OK
<data>

- não → HTTP Response
HTTP/1.1
304 Not Modified

Email

- Três componentes principais:
  - Agentes de usuário (MTA - Mail Transfer Agent) → Clientes de Email: ~~Thunderbird~~, mutt, Interface Gmail
  - Servidores de email → TCP 25 / Mensagens em ASCII / RFC 821
  - Protocolo SMTP (Simple Mail Transfer Protocol)



1:) S: 220 exim 3.2 hamburger.edu
C: HELO crepes-fr
S: 250 Hello crepes-fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender OK
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient OK
C: DATA
S: 354 Digite o email, terminando com '.'

/ /

C: Olah
C: Blabla
C: .
S: 250 Message accepted for delivery (muitas vezes há uma identificação única para a mensagem)
C: QUIT → Já emulamos o SMTP usando Telnet
S: 221 hamburger.edu closing connection

Para segurança → Utiliza-se PGP e GPG (Criptografia de chave pública/privada) para encriptar e assinar mensagem.

10/04

E-mail

RFC → 822 (Padrão para mensagens de texto enviadas por email) (ASCII)

- Define um cabeçalho → To:
  From:
  Subject:
  ...
- Define um corpo → Texto livre na codificação ASCII

Existem extensões para permitir outros formatos/conteúdos em uma mensagem de email:

- MIME (Multimedia mail extension) MIME-Version: 1.0 Content-Transfer-Encoding: base64 Content-Type: image/jpg
  RFC 2045 e 2056

Derivem extensões no cabeçalho de email para permitir conteúdo multimédia.

POP → RFC 1939 + GPG ou PGP
IMAP → RFC 1730 criptografia

Cliente / Servidor usando UDP

Demonstração do funcionamento.

Anonimização na Internet

→ Endereço IP de origem, porta de origem
⇓
Geolocalização
↳ pode ser usado para inferir o S.O.

→ Endereço IP de destino, porta de destino
⇓
identificar qual o destino (servidor) da comunicação
↳ pode inferir qual a aplicação sendo acessada

→ Serviço sendo acessado (no caso do HTTP, o URL)
→ Momento em que o serviço está sendo acessado

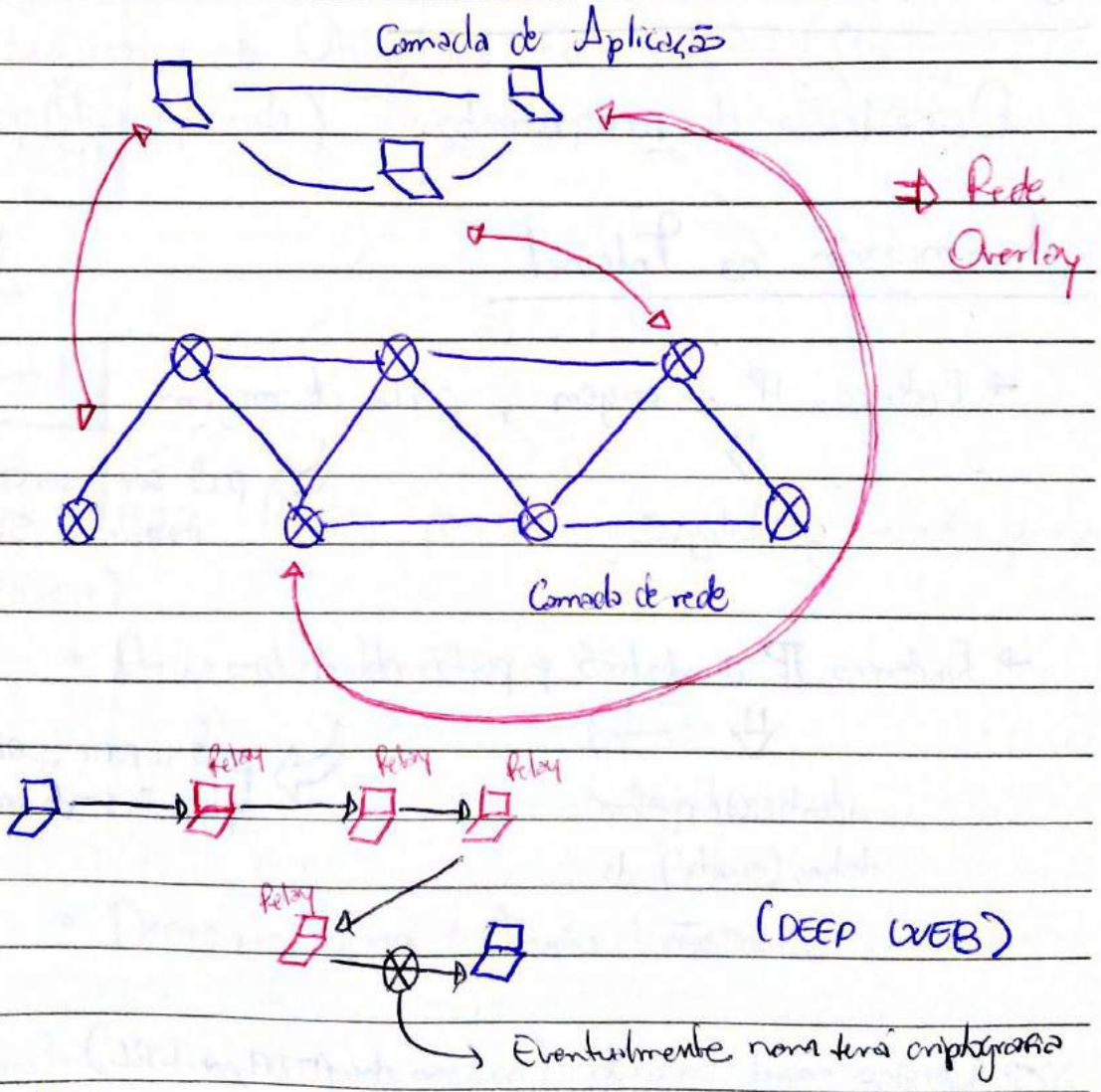
→ Proxys ajudam a "mascarar" algumas informações.
(Desde que o adm do proxy seja uma pessoa confiável)

/ /

→ Criptografia ajuda (como o HTTPS) mas algumas informações ainda ficam visíveis (IPs e Portas)

TOR (The Onion Router)

→ Objetivo: garantir comunicações anônimas na Internet



→ Todas as etapas são criptografadas !!!

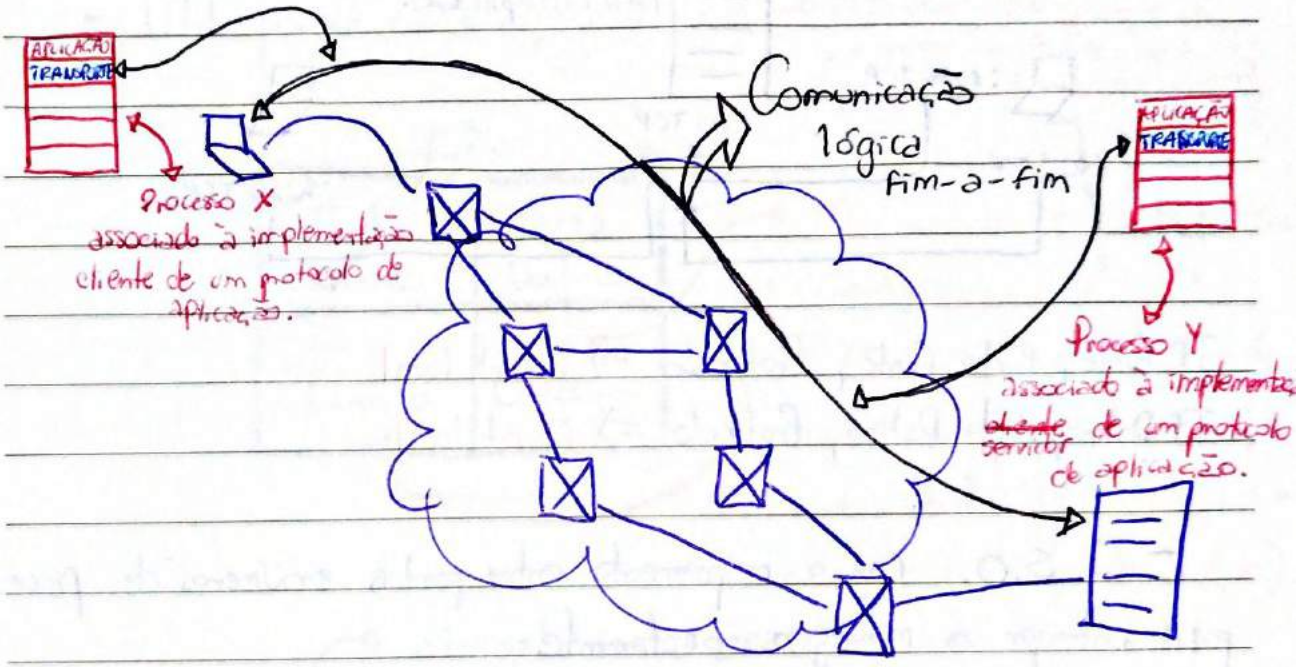
tilibra

tor project.org

12/04

Camada de Transporte

- Fornece uma comunicação lógica entre processos de aplicação em diferentes hosts.

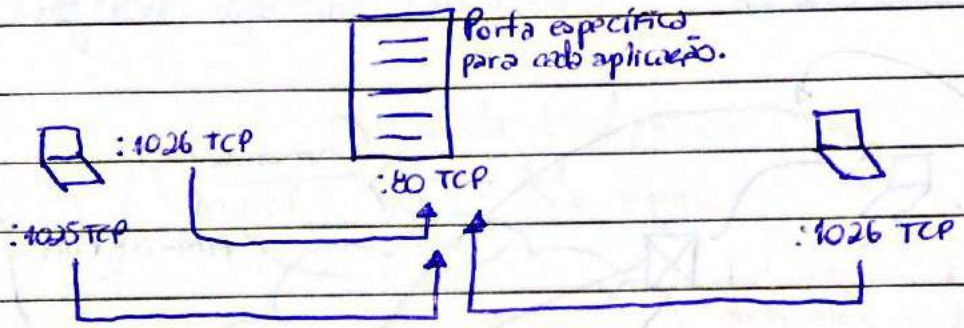


- Os protocolos de camada de transporte são executados nos hosts onde as aplicações são executadas (o próprio kernel do S.O. já traz os protocolos implementados).
- Lado emissor quebra as mensagens de aplicação em segmentos (Um segmento é a menor unidade na camada de transporte) e envia para a camada de rede (desce na pilha de protocolo).
- Lado receptor remonta os segmentos em mensagem e passa para a camada de aplicação (sobe na pilha de protocolo).
- Há dois protocolos disponíveis → UDP → não há garantias ("melhor esforço")
  → TCP → confiável, garante entrega, controle de fluxo, controle de congestionamento

/ /

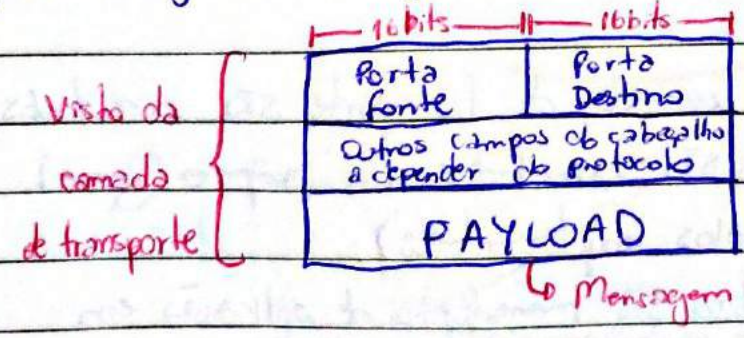
Não há garantia de atraso máximo e nem de largura de banda mínima na camada de transporte, independente do protocolo.

- multiplexação / demultiplexação de segmento
  - é necessário diferenciar os fluxos de mensagens pelas portas



IP fonte, Porta fonte, Protocolo ⇒ Socket local
IP Destino, Porta Destino, Protocolo ⇒ Socket local

- O S.O. faz o mapeamento entre portas enumeras de processo para entregar as mensagens corretamente



Protocolo UDP

- UDP: User Datagram Protocol
  RFC 768

- Protocolo sem garantia. Não vai tomar nenhuma ação se:
  - Segmentos forem perdidos
  - Segmentos forem entregues fora de ordem

} a camada de aplicação precisaria implementar algo para tratar desses casos

- Sem conexão
  - Não há apresentação entre transmissor e receptor.
  - Cada segmento é tratado de forma independente.
- Útil porque ele é rápido e leve → DNS usa UDP porque é um protocolo de aplicação que precisa de um retorno rápido

→ Mecanismo mínimo p/ detecção de erro

→ Tamanho máximo é 2^{16} ≅ 65.535 (aproximadamente porque tem que retirar os bytes do cabeçalho UDP - 8 Bytes - e o cabeçalho do IP que são 20 Bytes)

Checksum
- → obrigatório no IPv6
- existe para detectar erros de bits trocados
- → opcional no IPv4

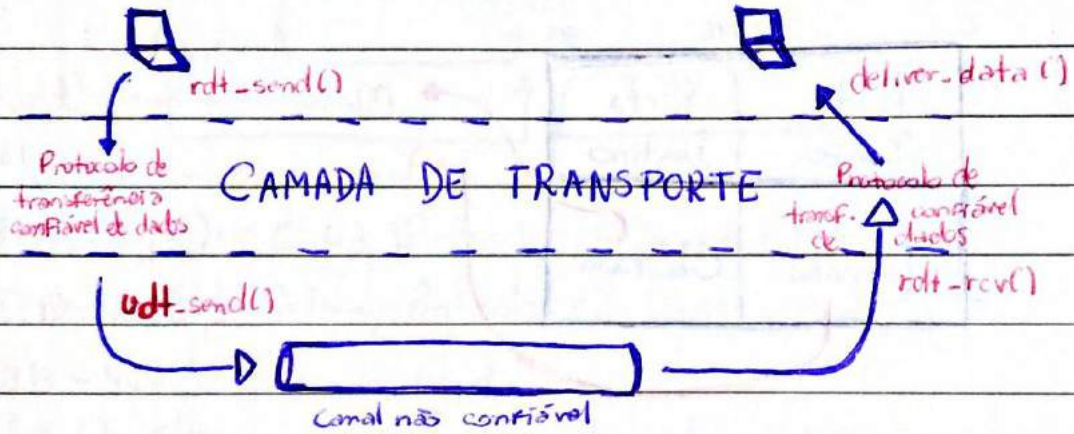
- O que é feito no transmissor?
  - Trata o conteúdo do segmento como sequências de inteiros de 16 b.

  - Soma o conteúdo de todos os números de 16 bits
  - Tira o complemento de 1 do resultado
  - Coloca o complemento de 1 no campo checksum do cabeçalho

/ /

- O que é feito no receptor?
  - Soma dos grupos de 16 bits junto com o checksum
  - Se o valor dor diferente de 11...11 é porque algo foi trocado.

RDT → Reliable Data Transfer



UDT → Unreliable Data Transfer

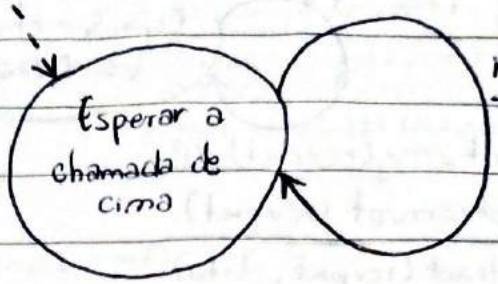
Vamos visualizar as RDTs como máquinas de estado

RDT 1.0

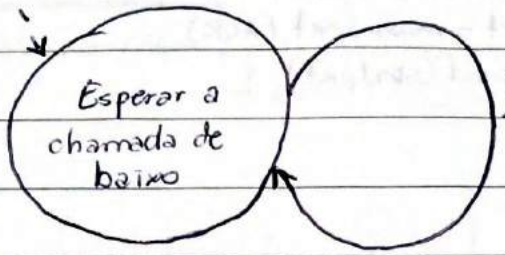
- Canal completamente confiável
  - Não há perdas
  - Não há erros
- Igual ao funcionamento do UDP

/ /

Máquina de estado do lado do remetente



Máquina de estado do lado do destinatário



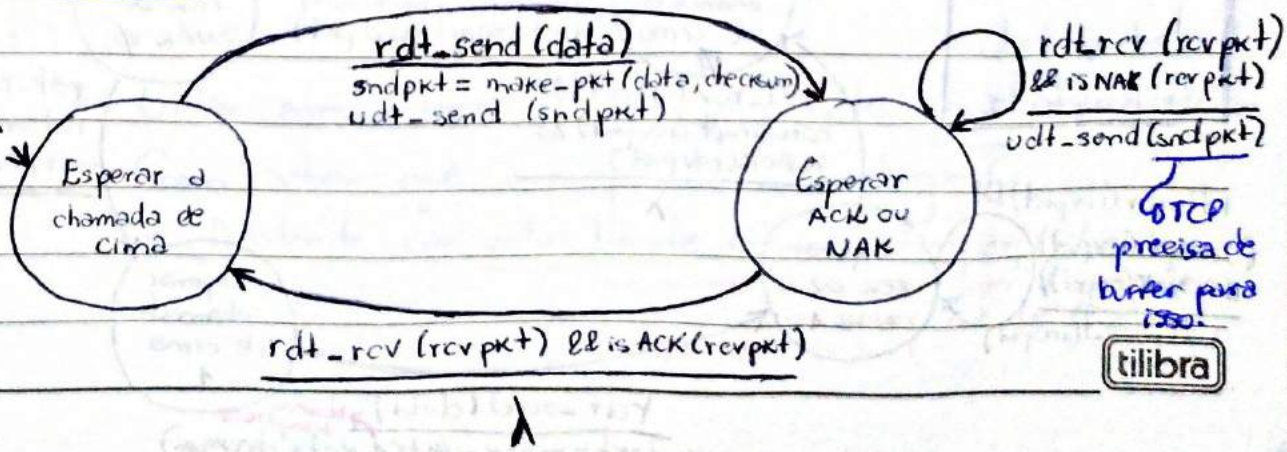
RDT 2.0

- Canal com erro de bit
  - Todos os pacotes chegam mas algum pode chegar com conteúdo corrompido.
- Como verificar que há erros? → Ex.: checksum, etc.
- Como se recuperar de erros? → Ex.: Automatic Repeat reQuest (ARQ)

TCP tem ACK:
- ACK: Receptor avisa ao transmissor que o pacote foi recebido corretamente.
- NAK: Receptor avisa ao transmissor que o pacote tem erros.

- ACK → Transmissor recebe o ACK ⇒ envia o próximo pacote
- NAK → Transmissor recebe o NAK ⇒ re-envia o pacote anterior

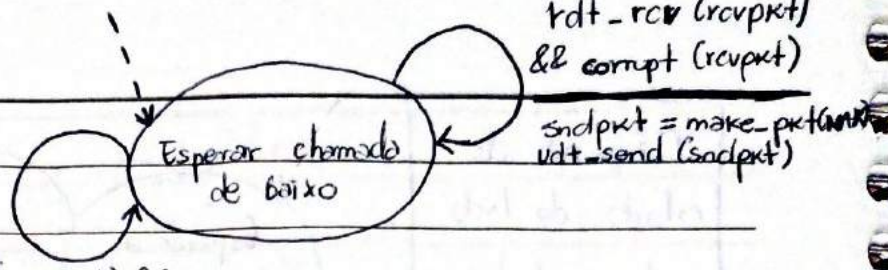
Máquina de estado do lado do remetente



↳ TCP precisa de buffer para isso.

/ /

Máquina de estado do lado do destinatário



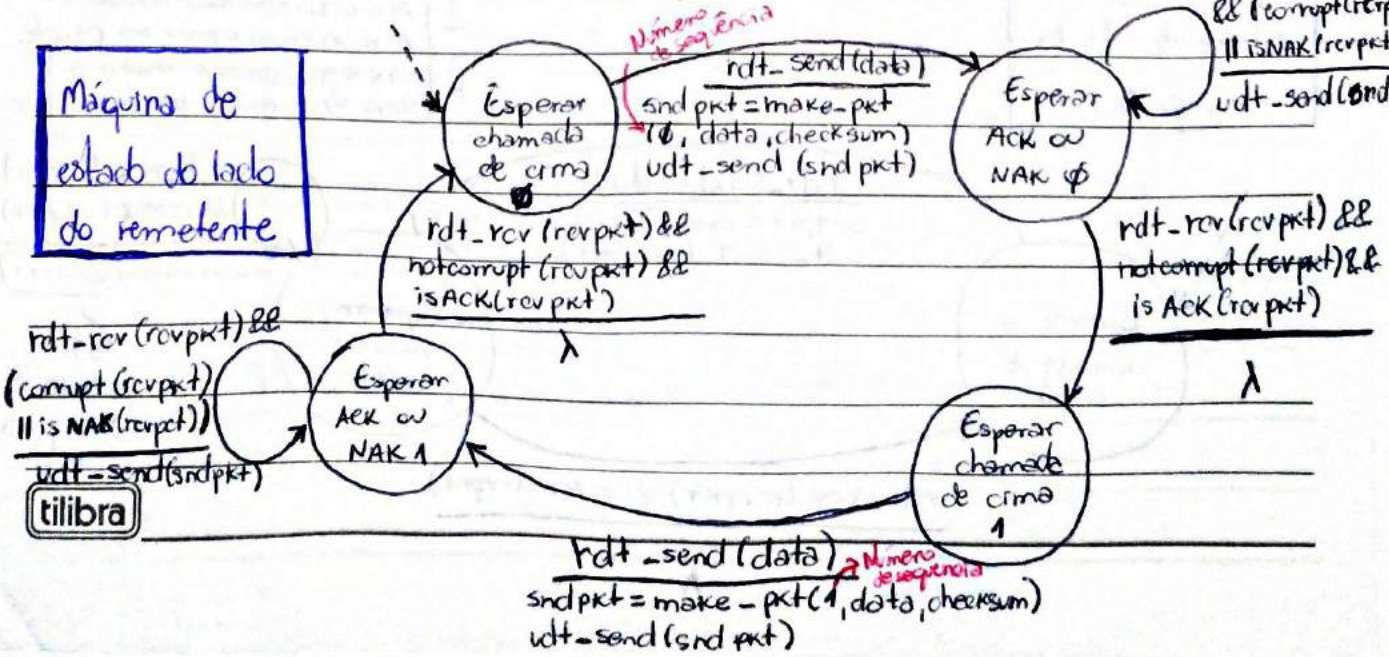
rdt_rcv (rcvpkt) &&
notcorrupt (rcvpkt)
extract (rcvpkt, data)
deliver_data (data)
sndpkt = make_pkt (ACK)
rdt_send (sndpkt)

RDT 2.1

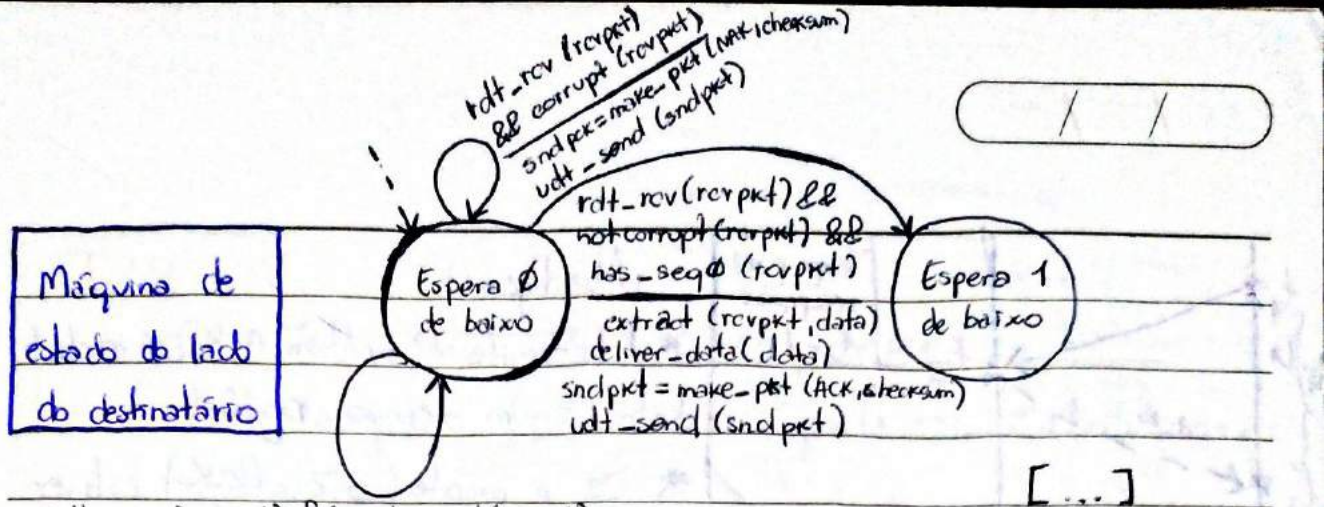
- No RDT 2.0 não estamos considerando que qualquer pacote pode ter erros.
- Erros nos ACK/NAKs precisam ser tratados e nesse caso, se for notado erro nesse reconhecimento, reenvia o pacote de dados.
- Como saber se o pacote é novo ou repetido?
  Usando números de seqüência (TCP implementa isso)
  No nosso caso, vamos usar apenas 0 e 1 para identificar 2 pacotes.

Do forma que estamos implementando o protocolo, ele é um protocolo PARE E ESPERE.

Máquina de estado do lado do remetente

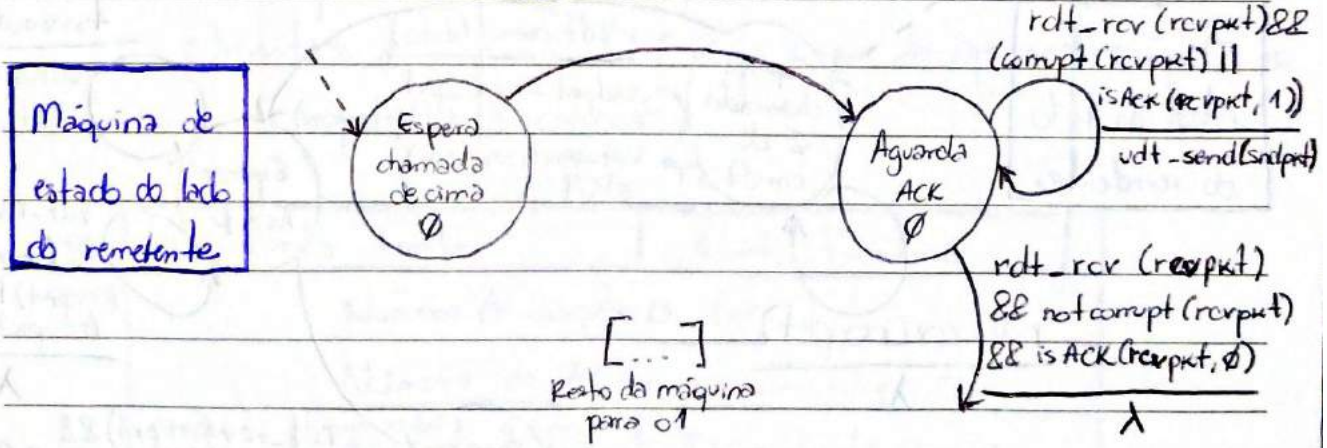


tilibra



Também implementado no TCP para "mostrar" que a rede não está tão ruim assim.

- RDT 2.2 (protocolo sem NAK)
  - O ACK explicita o número de sequência do pacote sendo reconhecido

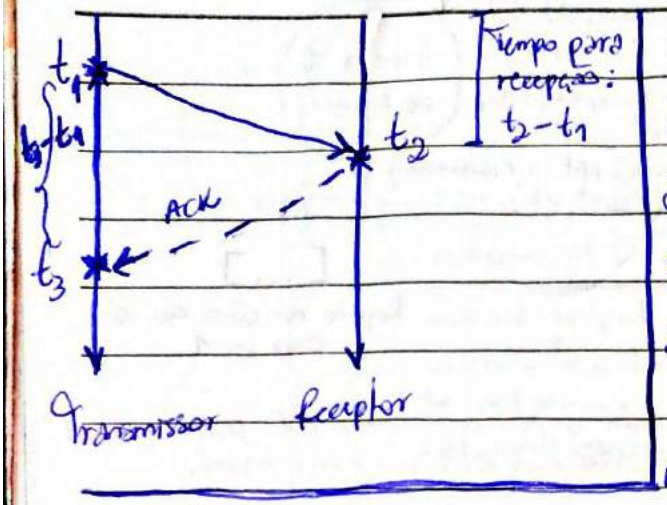


Máquina de estado do lado destinatário é similar.

RDT 3.0

- Lidar com o resto de que o pacote pode nunca chegar
- Como saber que um pacote não chegou?
  - Adotando um valor limite em um temporizador
    - Esse valor deveria ser definido com experimentos considerando nós bem distantes em termos de RTT.

/ /



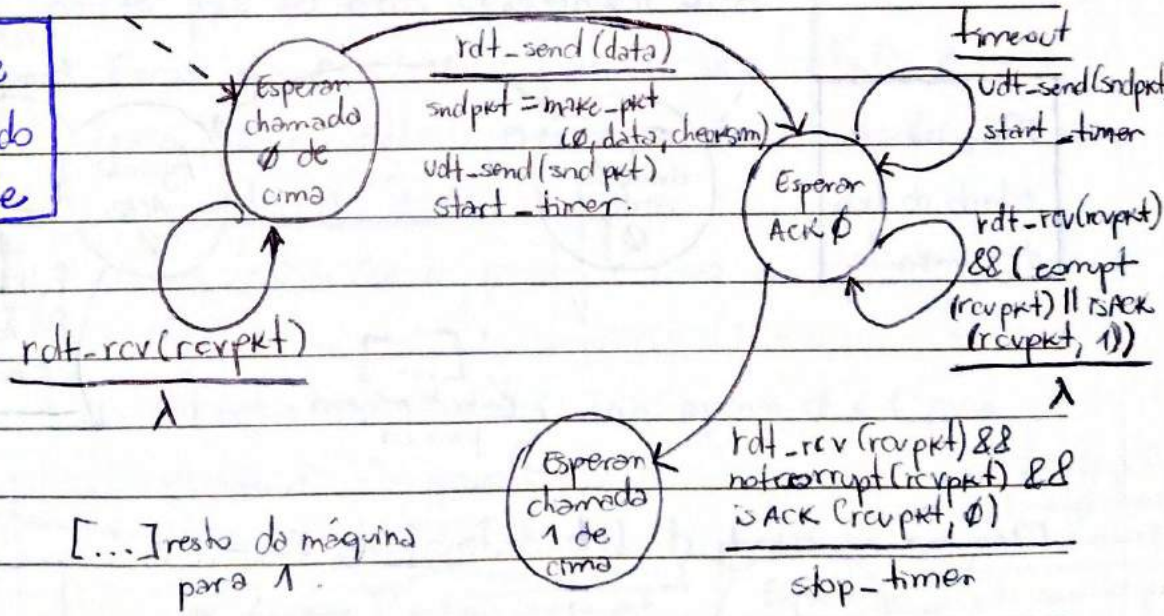
Tempo para recepção: $t_2 - t_1$

Abordagem:

* Retransmite se nenhum ACK foi recebido dentro de um tempo razoável.
* Se o pacote (ou o ACK) estiver apenas atrasado:
  - Retransmissão será duplicata, mas o número de sequência já trata disso.

* Receptor deve especificar o nº de sequência do pacote sendo reconhecido. mais variáveis no protocolo
* Exige um temporizador decrescente.

Máquina de estado do lado do remetente



[...] resto da máquina para 1.

Máquina de estado do lado do destinatário é similar.

/ /

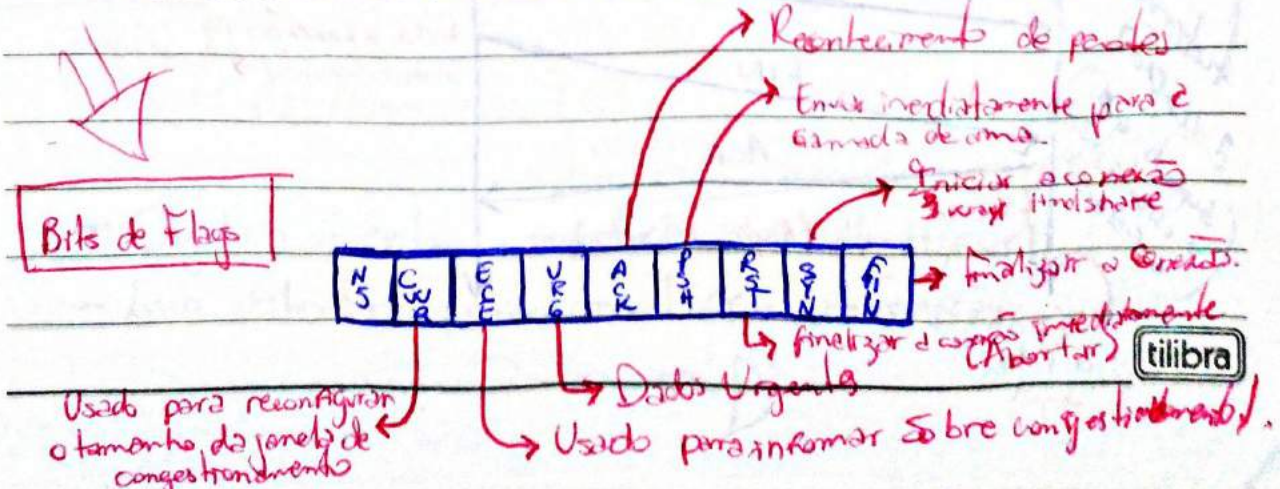
TCP

- RFC 793, 1122, 1323, 2018, 2581
- Ponto-a-ponto (porta fonte e porta destino identificam as aplicações)
  - 1 transmissor } unicast
  - 1 receptor }
- Confiável: envio de um fluxo de bytes sequencial
- "Pipelined": transmissão de vários pacotes sem confirmação
- Controle de fluxo e controle de congestionamento evitam inundação da outra ponta da rede
- full duplex (pacotes trafegam nos 2 sentidos ao mesmo tempo)
- Orientado a conexão (há uma fase de apresentação entre as duas pontas da conexão) 3 way handshake

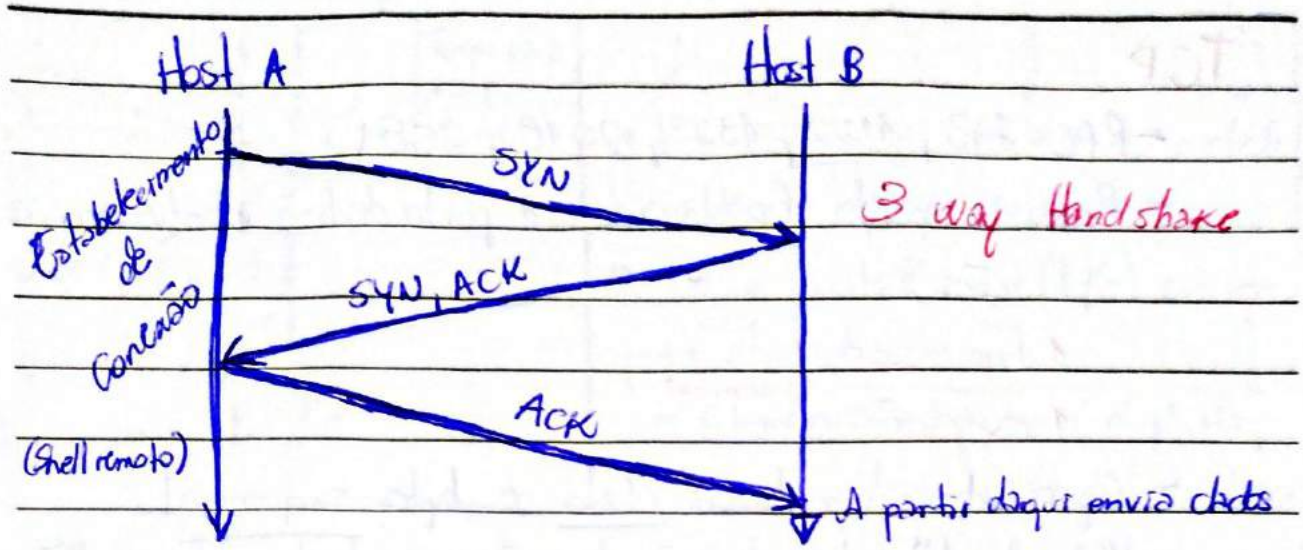
Cabeçalho TCP

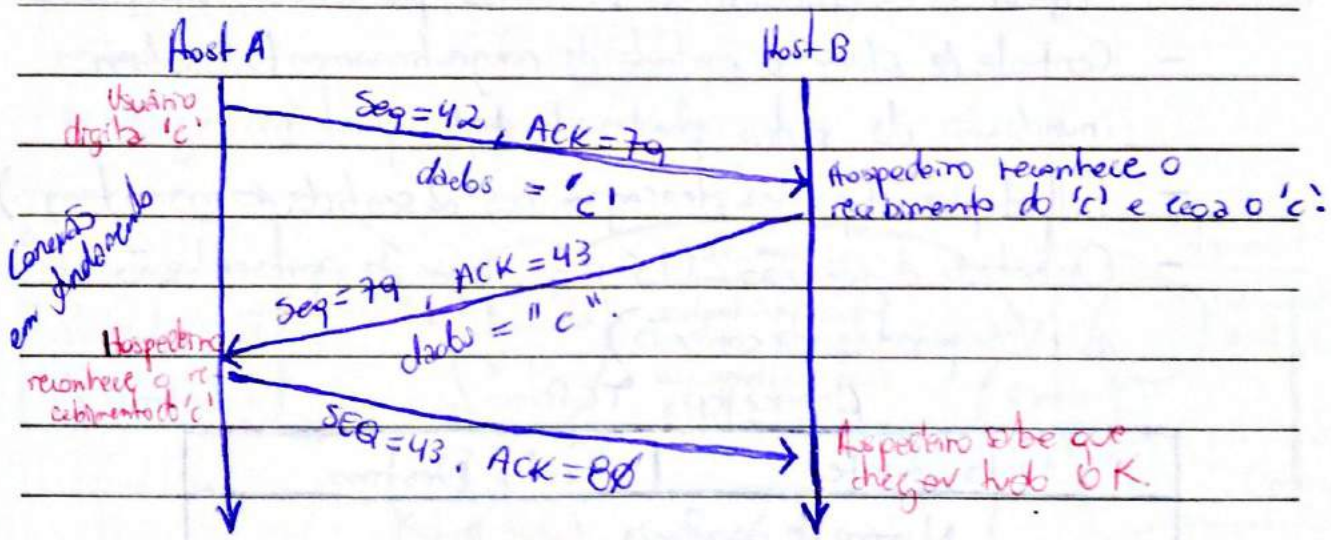
Porta fonte	Porta Destino
Número de sequência (nº do byte)	
Número do ACK (próximo número de sequência)	
Offset de dados | Reservado 000 | 9 bits de flags	Tamanho da janela
checksum	Ponteiro para dados urgentes (URG)
Opções, se data offset > 5	

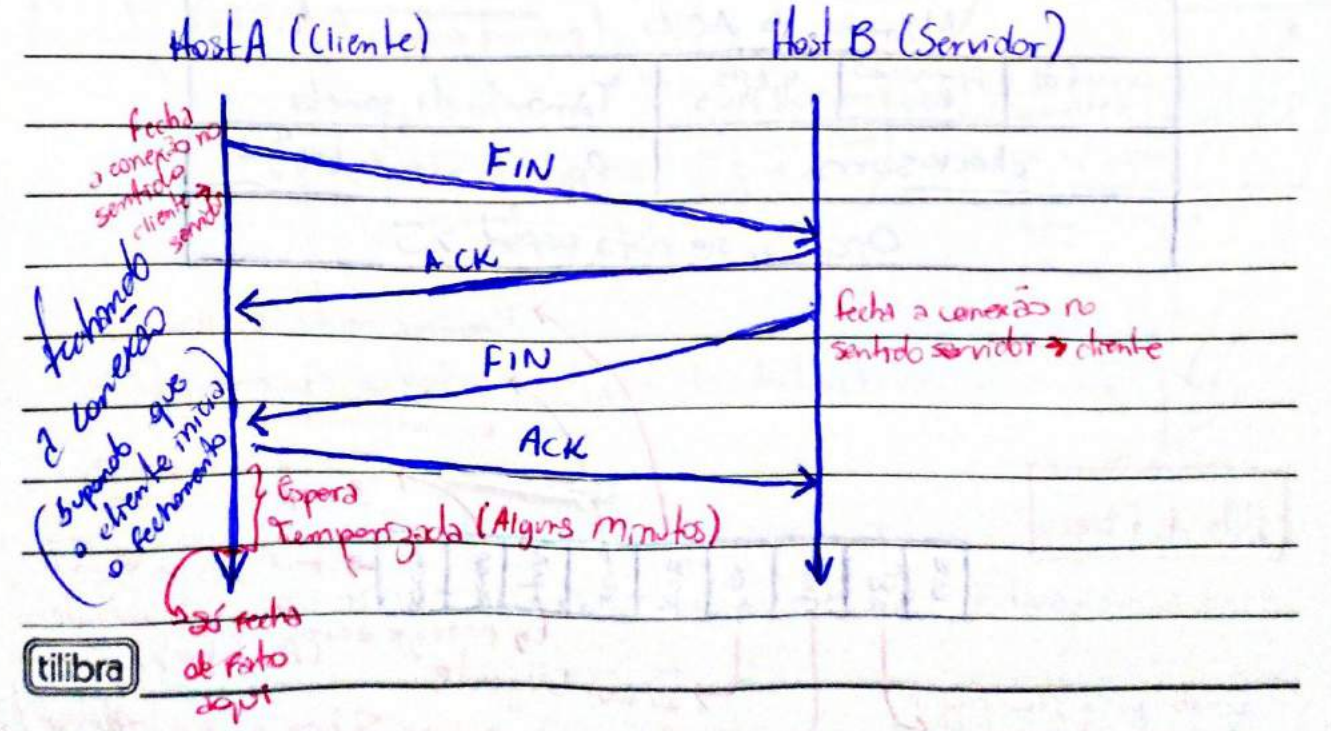
Bits de Flags



/ /

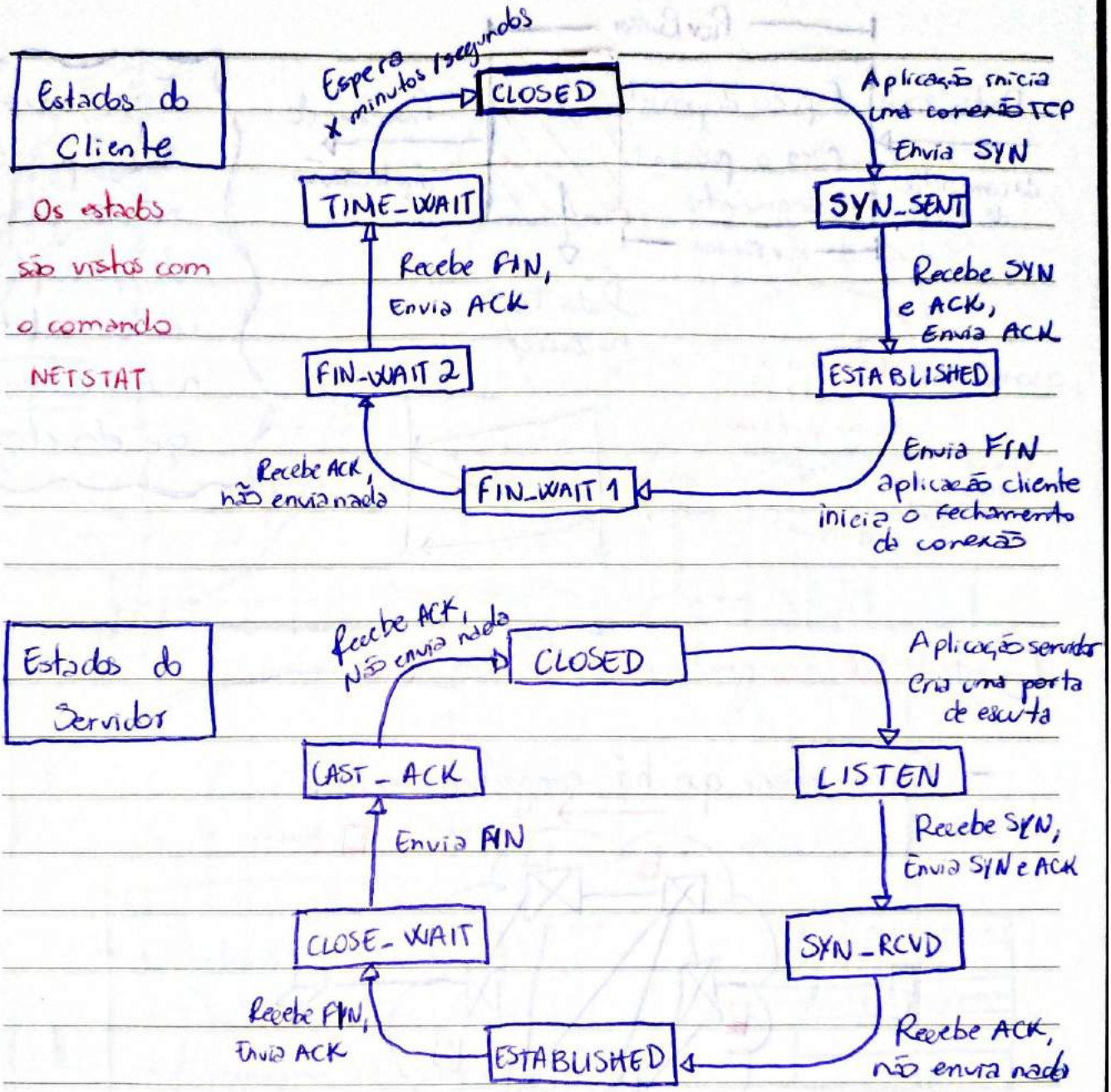

Host A
Host B
Estabelecimento de Conexão
SYN
SYN, ACK
ACK
3 way Handshake
(Shell remoto)
A partir daqui envia dados


Host A
Host B
Usuário digita 'c'
Seq=42, ACK=79
dados = 'c'
Host B reconhece o recebimento do 'c' e ecoa o 'c'
Conexão em andamento
Seq=79, ACK=43
dado = "c".
Host A reconhece o recebimento do 'c'
SEQ=43, ACK=80
Host B sabe que chegou tudo OK.


Host A (Cliente)
Host B (Servidor)
Fecha a conexão no sentido cliente → servidor
FIN
ACK
Fecha a conexão no sentido servidor → cliente
Fechando a conexão
(Supondo que o cliente inicia o fechamento)
FIN
ACK
Espera Temporizada (Alguns minutos)
Só fecha de fato aqui

tilibra

/ /



08/05

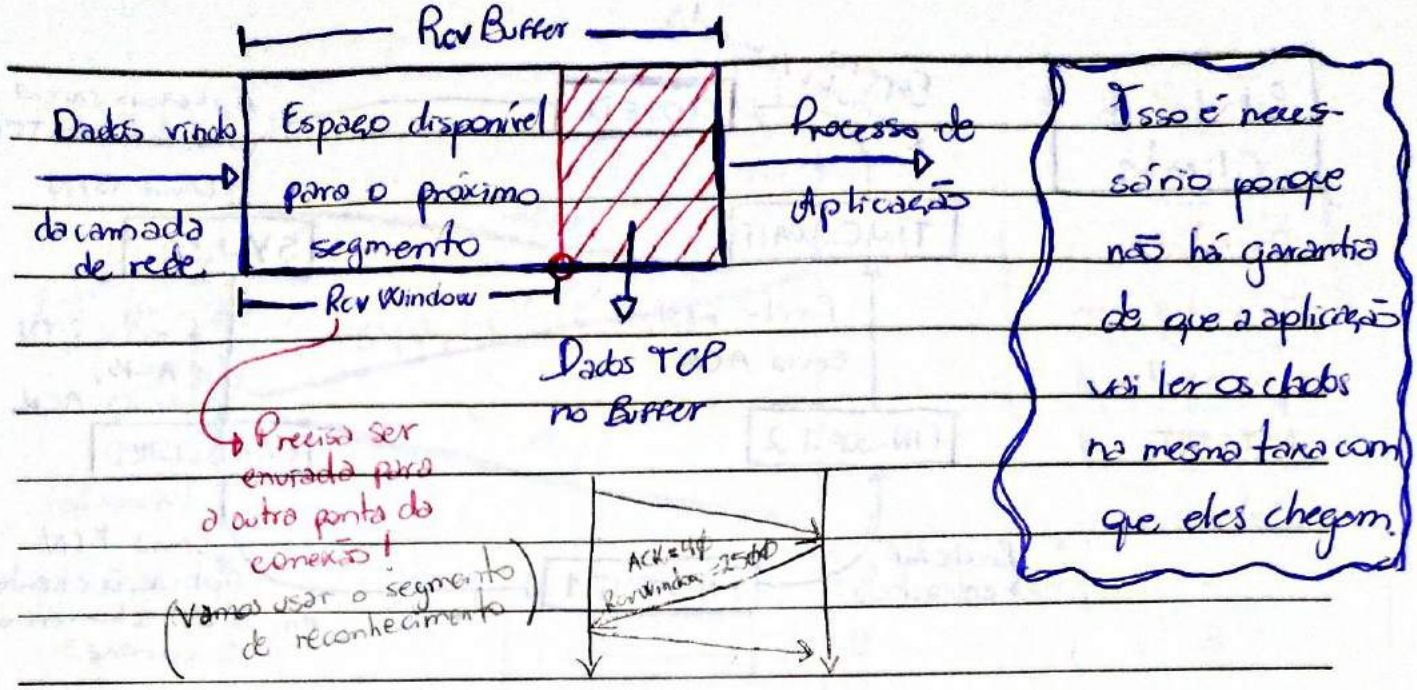
Controle de fluxo do TCP

Não causar estouro do buffer de recepção

O dono do IP destino do pacote

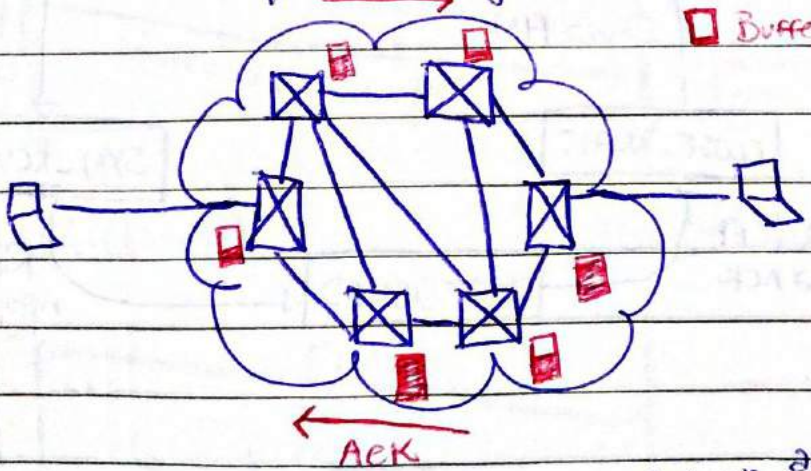
- Objetivo é evitar a inundação do destinatário
- Lado destinatário possui um buffer de recepção

/ /



Controle de congestionamento do TCP

- Como inferir que há congestionamento?

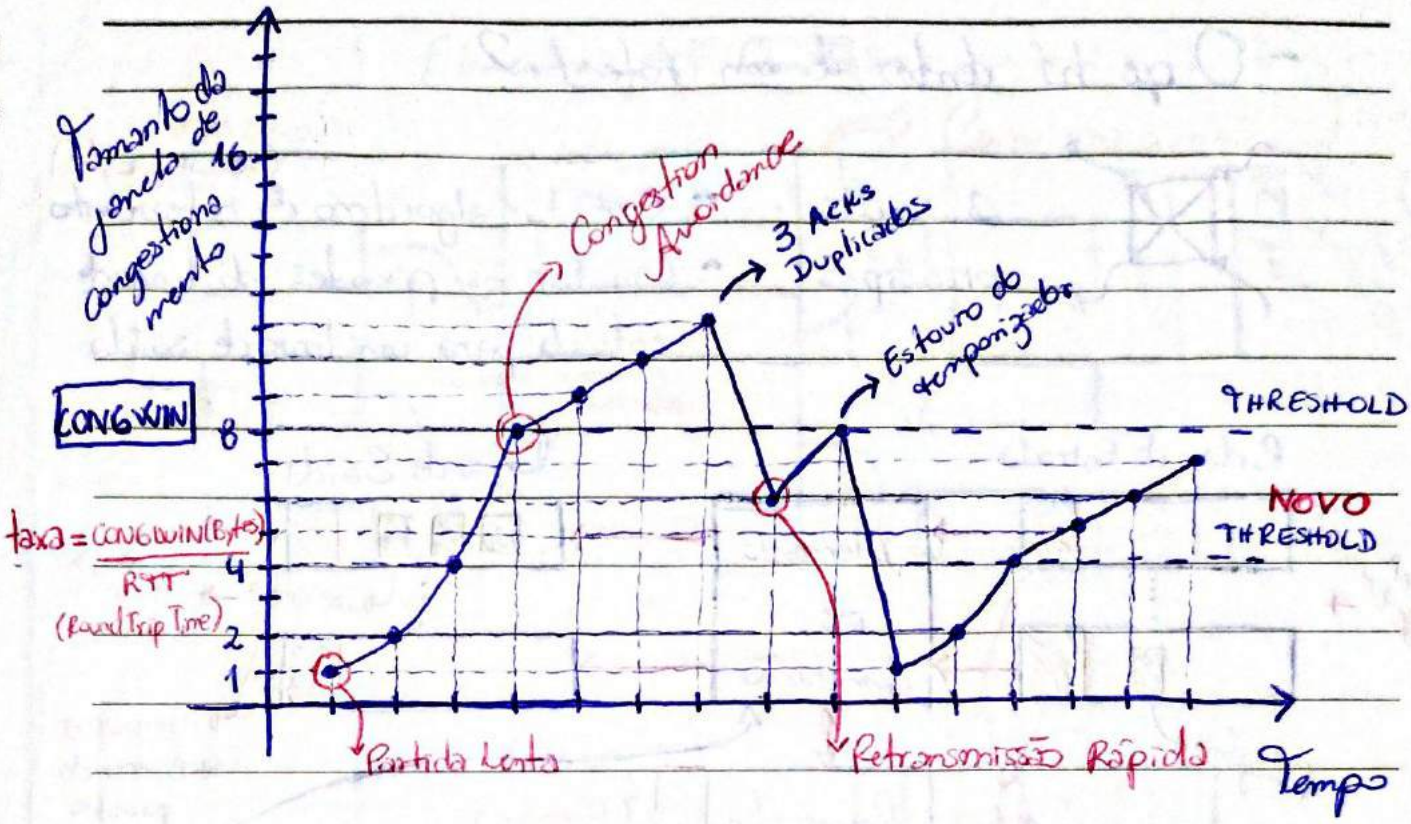


- Preocupações:
  - Segmento Perdido (Estouro do Temporizador) → Supõe que a rede está muito ruim
  - Segmento demora muito para chegar (3 ACKS Duplicados) → supõe que a rede está ruim mas não tão ruim assim.

→ Reduzir a janela de congestionamento para 1 e othreshold pela metade do threshold atual

↳ Reduzir a janela de congestionamento pela metade

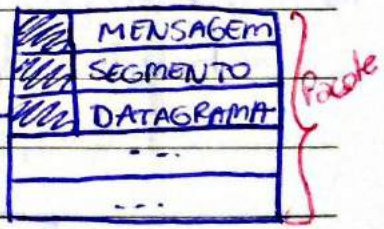
tilibra



Camada de Rede

→ Principalmente com o núcleo da rede

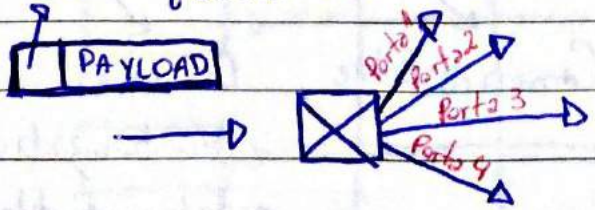
- Objetivos:
  - Identificação dos hosts → Endereço IP
  - Roteamento de datagramas



→ Baseado no endereço IP de destino e é realizado pelos roteadores que são configurados com tabelas de roteamento

→ Está presente no cabeçalho IP do pacote

Cabeçalho { Endereço IP de origem / Endereço IP de destino



/sbin/route -n

- Default Gateway
- Gateway Padrão
- Rota Padrão

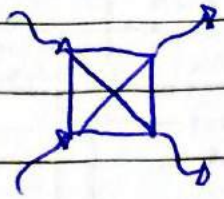
Endereço 1	Porta 1
Endereço 2	Porta 2
Endereço 3 / Endereço 4	Porta 3
*	Porta 4

Processo de enviar pacotes entre roteadores é ROTEAMENTO

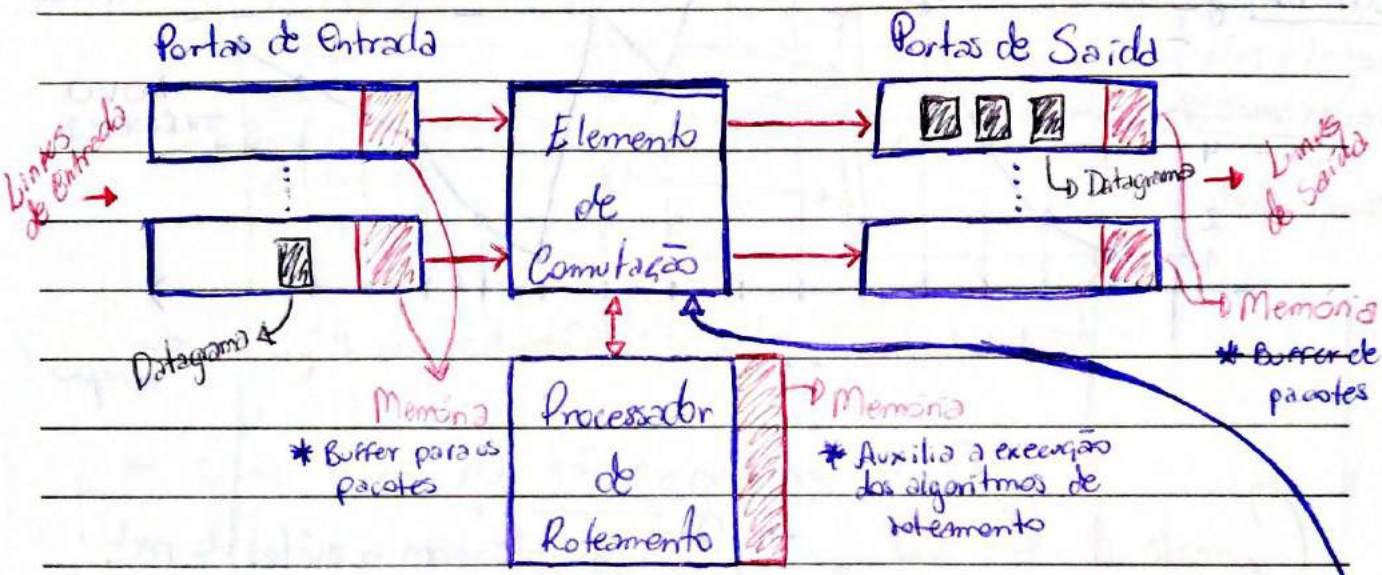
Processo de enviar um pacote de uma porta de entrada para uma porta de saída é COMUTAÇÃO

10/05/19

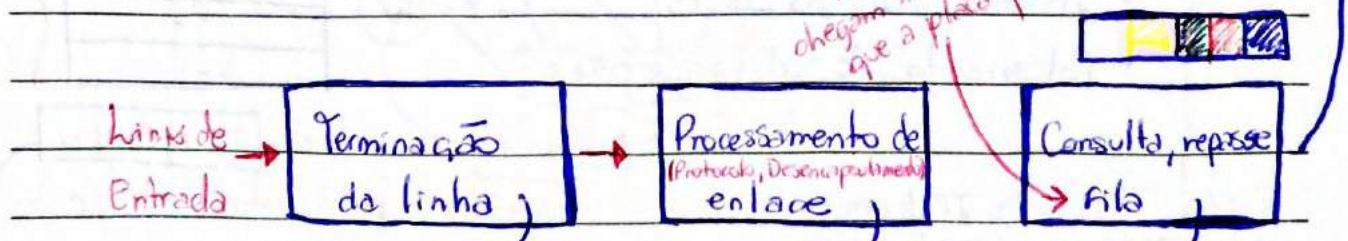
- O que há dentro de um roteador?



2 funções principais:
- Executar algoritmo de roteamento (rotear pacotes)
- Comutar os pacotes do link de entrada para um link de saída



Funções da porta de entrada

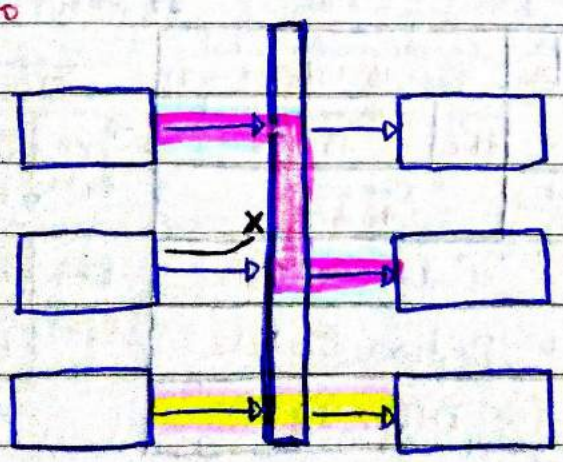


Elemento de Comutação

- memória

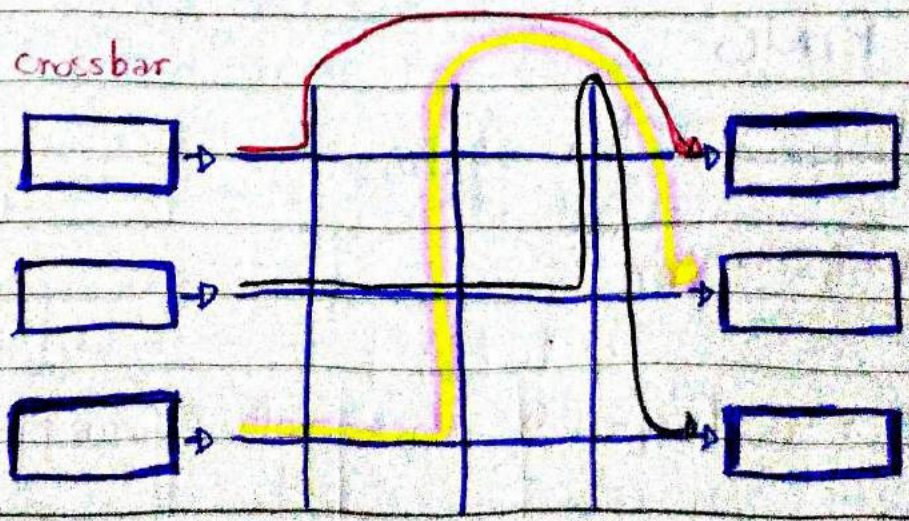
Gargalo na comunicação!!!

- barramento



Se as interconexões não se cruzam, é possível a troca de dados em paralelo.

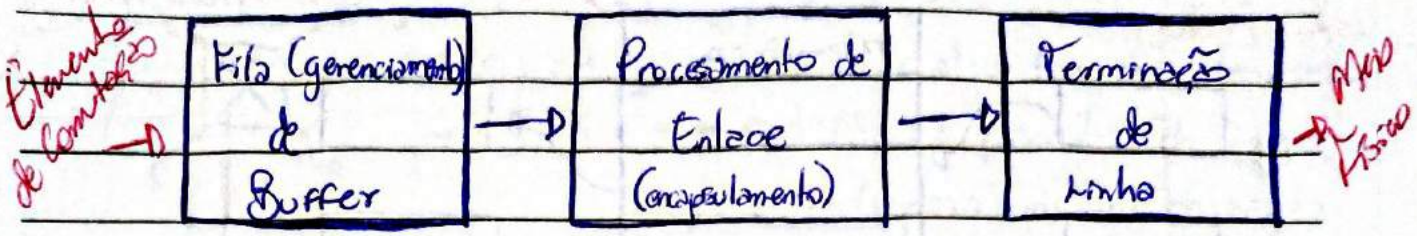
- crossbar



É o modo mais eficiente que permite ao roteador trabalhar na sua máxima capacidade nominal

/ /

Porta de Saída



- IP: o protocolo da Internet

Exemplos
- 000000 ⇒ "melhor esforço"
- 000010 ⇒ Tráfego de vídeo ao vivo (rápido)
- 010000 ⇒ Tráfego de vídeo em 4K (rápido e sem perda)

Permite diferenciação de serviços

Cabeçalho IP

Ø → 15 16 → 31 → bits

Versão 4 bits	IHL 4 bits (Tamanho do cabeçalho)	DSCP 6 bits	ECN 2 bits (Avisa que a rede está congestionada)	Comprimento Total 16 bits (do pacote)
Identificação 16 bits			Flags 3 bits	Fragment Offset 13 bits
Time to Live TTL 8 bits (valor decrementado a cada roteador)		Protocolo 8 bits (protocolo do próximo cabeçalho)		Checksum 16 bits
Endereço IP fonte 32 bits (Ex: 192.168.0.4)				
Endereço IP Destino 32 bits (Ex: 10.0.5.1)				
Opções (se IHL > 5)				

IPv4 / IPv6

Útil para pacotes fragmentados

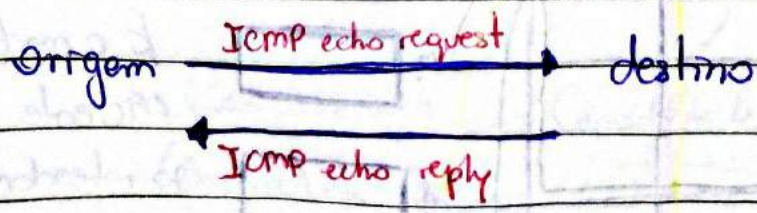
2000 Bytes

MTU → maximum transfer unit = 1500 Bytes

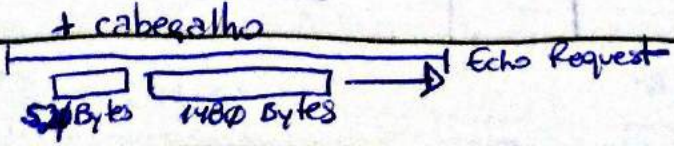
- 1500 Bytes | ID = 0x10 / Flags = " " / Fragment offset = 1501
- 1500 Bytes | ID = 0x10 / Flags = "more fragments" / Fragment offset = Ø

17/05

PING

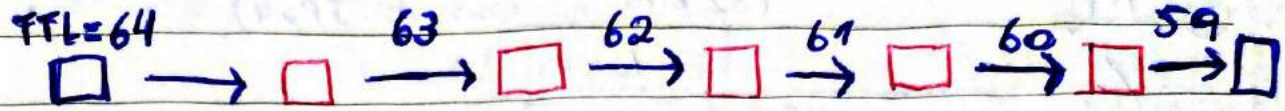


1 RTT = o tempo de ida e volta de um pacote ICMP

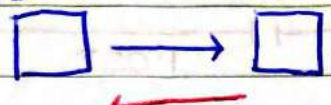


tilibra

TRACEROUTE

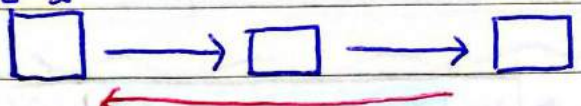


TTL = 1



Envia datagramas UDP para portas altas (5910 UDP)

TTL = 2



TTL = 3

[...]

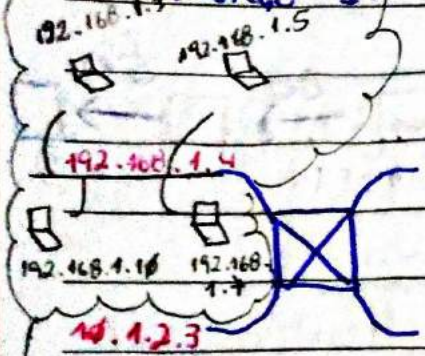
```
TTL = TTL - 1
If TTL == Ø:
    drop packet
    send ICMP TTL expired
```

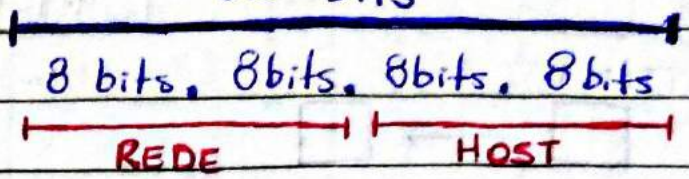
Endereçamento

/ /

CLASSE C

Endereço IP → identificador de interfaces de nós na Internet
(32 bits no caso de IPv4)





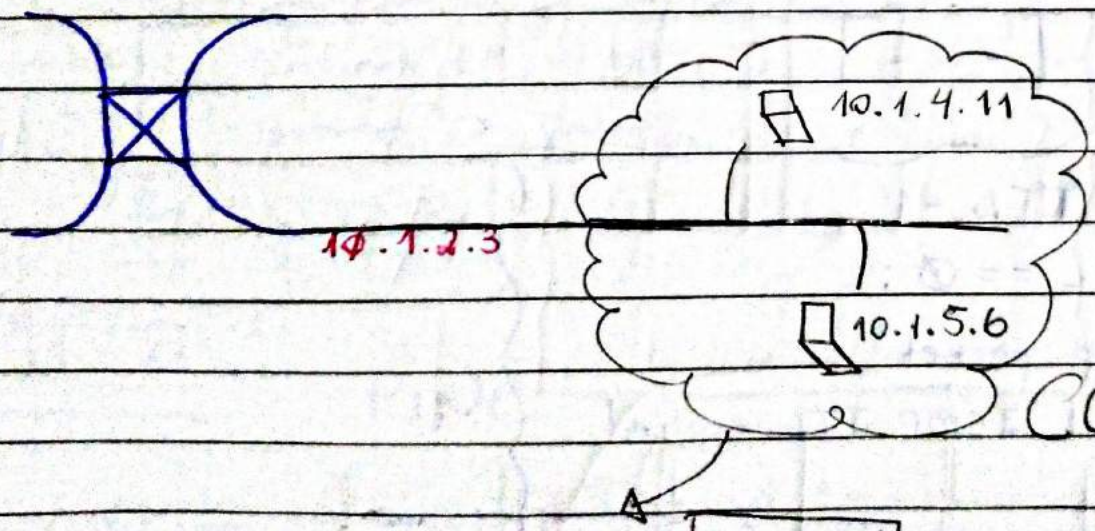
(Sub-rede)

Sub-rede → 192.168.1.0

Hosts → 192.168.1.1
a
192.168.1.254

Endereço 0 ⇒ Rede
Endereço 255 ⇒ Broadcast

[...]



Sub-rede → 10.1.0.0

Hosts → 10.1.0.1
a
10.1.255.254

/ /

CLASSE A → 10.0.0.0 Rede

10.0.0.1
a
10.255.255.254 Hosts

"Endereços IP Inválidos"

192.168.x.x
10.x.x.x
} serem usados em redes locais sem acesso à Internet

Máscara de rede ⇒ 10.4.0.0 / 255.255.0.0
10.4.0.0 / 16 Classe B

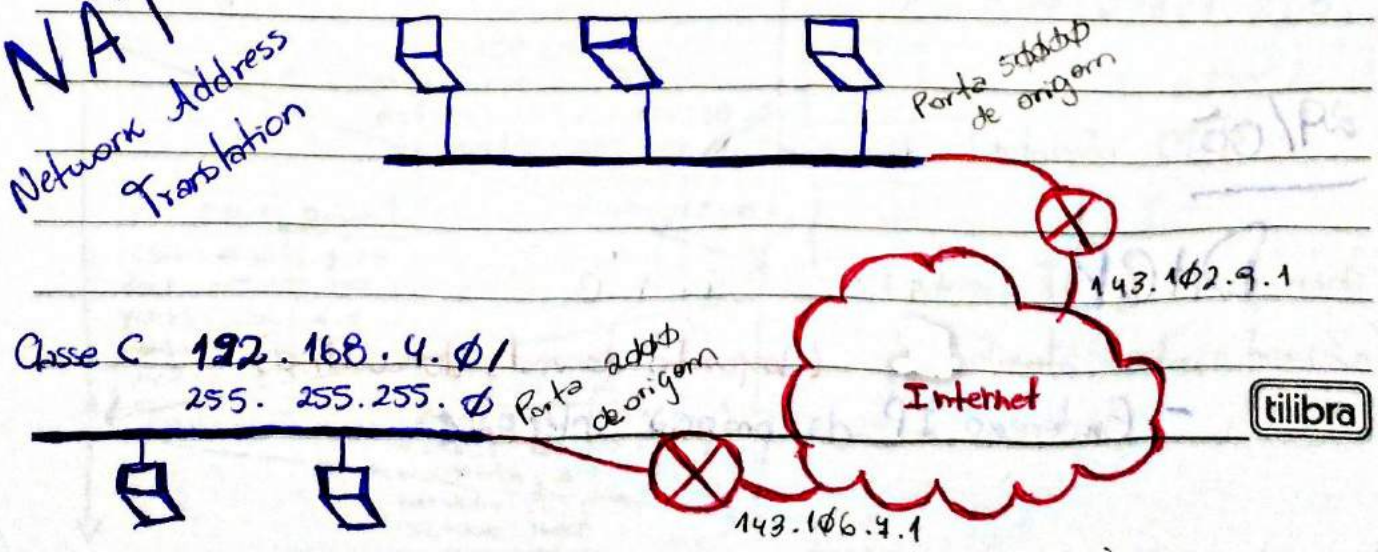
Endereçamento / Máscara

192.168..1.0 / 255.255.255.0
192.168.1.0 / 24 Classe C

10.0.0.0 / 255.0.0.0
10.0.0.0 / 8 Classe A

NAT
Network Address Translation

Classe A 10.0.0.0 / 255.0.0.0



/ /

Roteador NAT

Local Area Network

Wide Area Network

Interface LAN	Interface WAN
10.0.0.1/50000	143.106.7.1/60000
10.0.0.2/20000	143.106.7.1/30000

Ex.: Rede interna tem 2 máquinas 10.0.0.1 e 10.0.0.2 e elas acessam respectivamente HTTP para o google e facebook

ICMP → Internet Control Message Protocol

- Usado para troca de informações de controle da camada de rede
- Relatar erros (TTL expirado! Porta ou host inalcançáveis!)
- Relatar estatísticas (Ping / Traceroute)
- Mensagens ICMP são identificadas por 2 números { Tipo / Código

	Tipo	Código	Descrição	
PING	8	0	echo request (ping)	→ Se for implementar um ping
	0	0	echo reply (pong)	→ Se for implementar resposta ao ping no SO
TRACEROUTE	11	0	TTL expired	→ Se for implementar um SO para roteador
	3	3	destination port unreachable	→ Se for implementar num SO

29/05

DHCP

(do ponto de vista do usuário)

- Endereço IP da própria interface

- Máscara da classe da rede
- Endereço IP do roteador padrão
- Endereços IP dos servidores DNS da rede

Dynamic Host Configuration Protocol

- Servidor DHCP na rede é responsável por configurar as máquinas 192.168.1.1 - 192.168.1.50 (ex.)
- Centraliza os "empréstimos" de endereços IP para todos os hosts

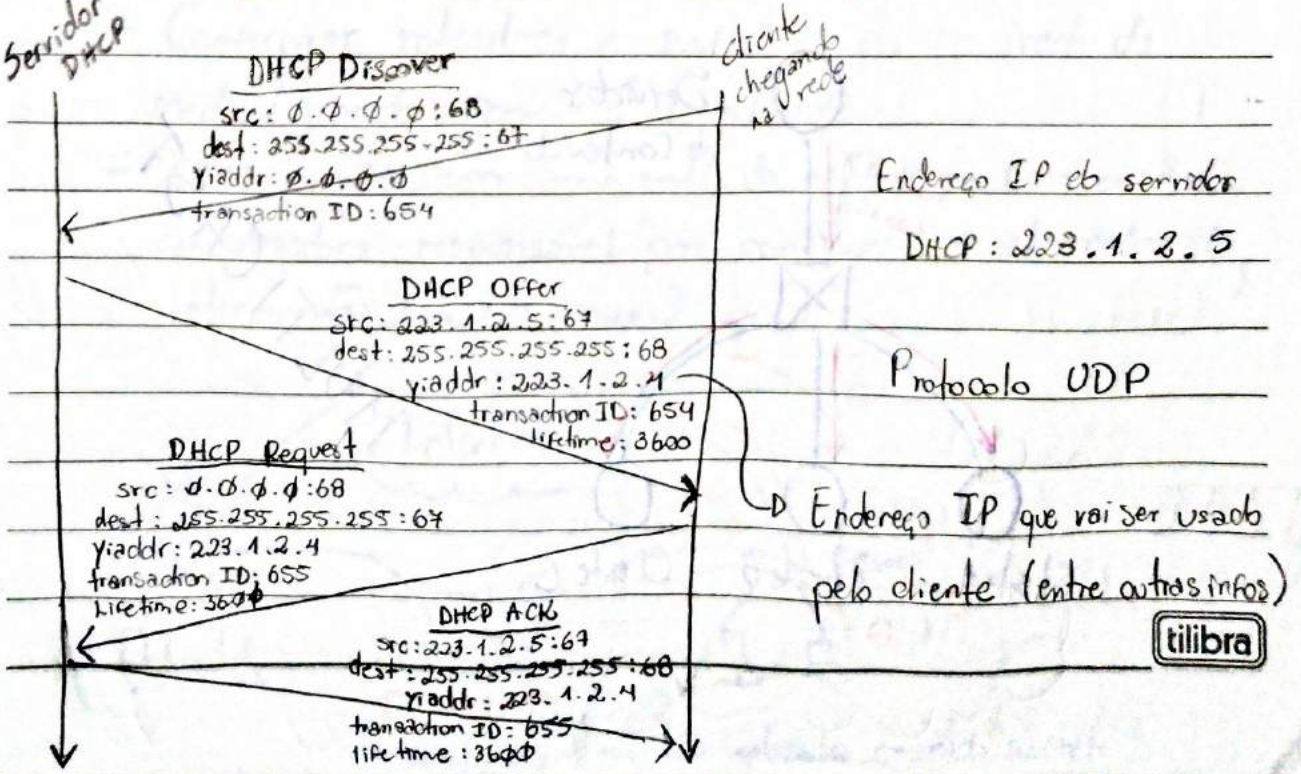
Endereço IP	Máquina	Tempo
1	X	3000
2	Y	3000
1	Z	3000
...	...	...
...	...	...

Informação de camada de enlace

Como identificar as máquinas?

Através do endereço de HW da placa de rede (Endereço MAC)

Essa tabela é gerenciada internamente pelo servidor DHCP



Endereço IP do servidor DHCP: 223.1.2.5

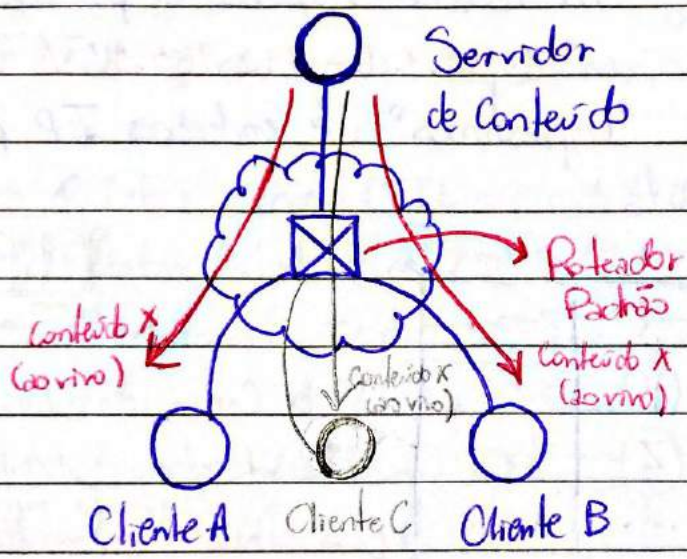
Protocolo UDP

Endereço IP que vai ser usado pelo cliente (entre outras infos)

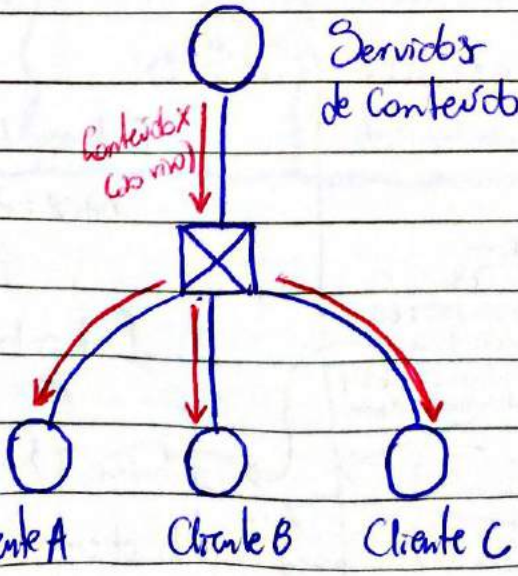
/ /

- Ao estourar o lifetime, o servidor DHCP renova o IP do cliente (caso ele queira). Se não houver confirmação na renovação, aquele endereço volta para o "pool" de endereços disponíveis.

Multicast



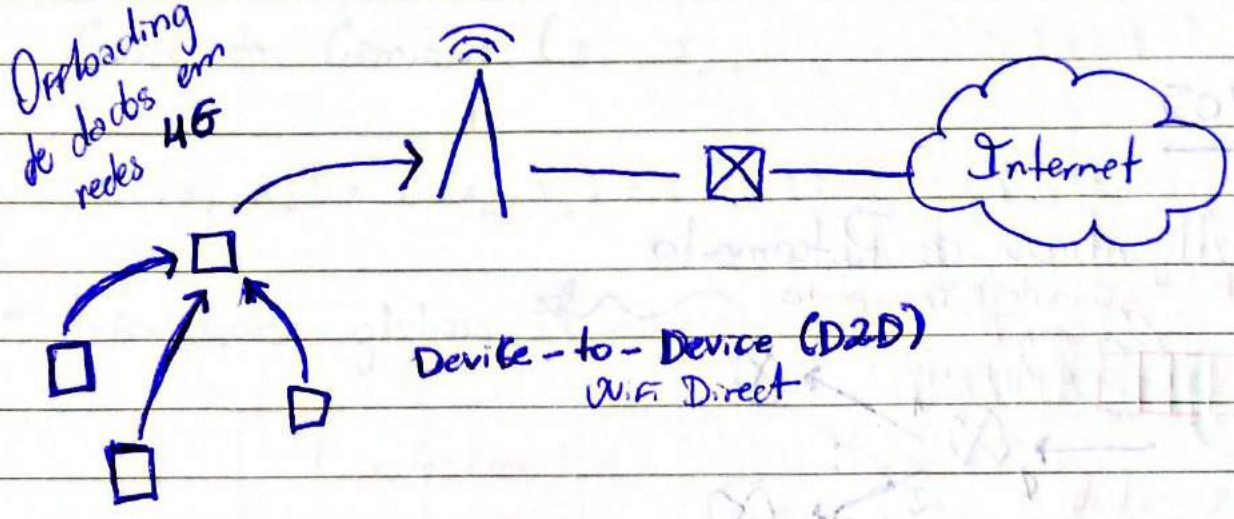
UNICAST



MULTICAST

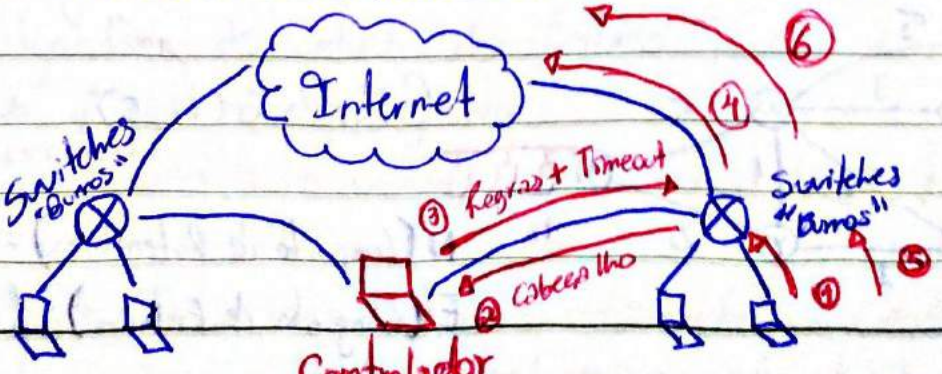
Restrições para funcionar:

- Roteadores precisam suportar o mesmo protocolo multicast
- Só funciona para UDP
- Exige na maioria das vezes (grupo multicast) configuração manual



Redes Definidas por Software (SDN)

- Configurar roteadores e switches de uma rede de médio/grande porte é complicado
- SDN propõe um ambiente de rede com uma camada centralizadora responsável por configurar os elementos de interconexão (middle boxes)



↳ Tem as regras que são aplicadas em cada switch

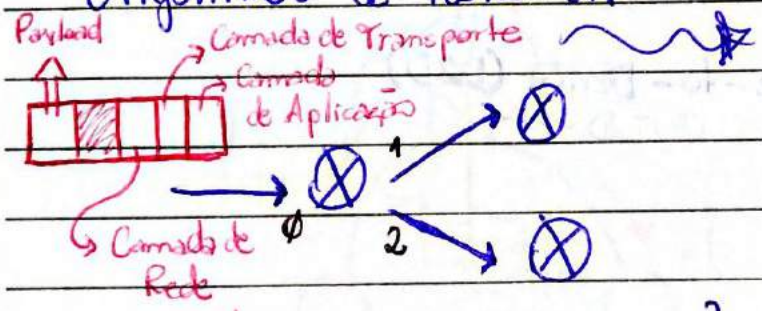
/ /

Principais características:

- A inteligência da rede está centralizada e é feita via software
- O protocolo entre controlador e switches é "padronizado" (p. ex. OpenFlow)
- É uma rede em que o plano de dados é separado do plano de controle

31/05

Algoritmos de Roteamento

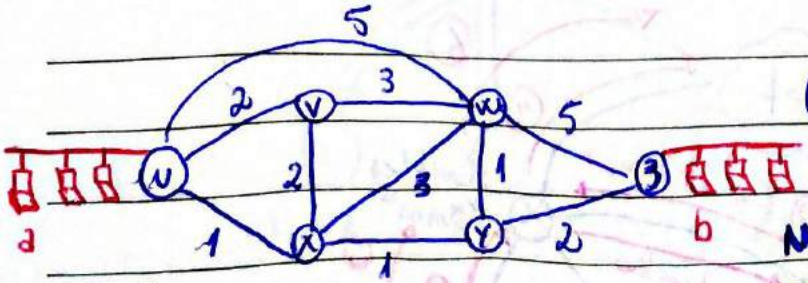


(Endereço fonte e destino) →
- ϕ1ϕϕ → Interface 1
- ϕ1ϕ1 → Interface 1
- ϕ111 → Interface 2
- 1ϕϕ1 → Interface 2
- * → Interface 2

} Tabela de Roteamento

- Roteamento ⇒ Tentando encontrar o menor caminho entre 2 nós na Internet.
  - ↳ Nem sempre está associado à distância.

↳ Grafos ⇒ Vamos buscar o menor caminho em um grafo



Grafo G(N, E) (nós; enlaces)

N (Conjunto de Roteadores) = {u, v, w, x, y, z}

E (Conjunto de Enlaces) = {uv, ux, vx, ...}

/ /

- Pesos das arestas auxiliam a busca pelo menor caminho.
  - ATRASO
  - CUSTO FINANCEIRO
  - DISTÂNCIA FÍSICA
  - INVERSO DA LARGURA DE BANDA
  - CONGESTIONAMENTO (% de perda nos enlaces)

Custo do Caminho $(x_1, x_2, \ldots, x_n)$

$$=$$

$$c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{n-1}, x_n)$$

- Informações globais ou descentralizadas
  - ↳ Todos os roteadores tem uma visão geral da rede (topologia) — ALGORITMOS DE ESTADO DE ENLACE
  - ↳ Roteadores só conhecem informações dos vizinhos e dos enlaces para esses vizinhos — ALGORITMOS DE VETOR DE DISTÂNCIA

- Atualização das Informações
  - Algoritmos Estáticos → Rotas mudam lentamente ao longo do tempo
  - Algoritmos Dinâmicos → Rotas mudam mais rápido

- Algoritmo de Estado de Enlace
  - * Algoritmo de Dijkstra
    - ↳ Computa o caminho de menor custo de um nó (fonte) para todos os outros nós.
    - ↳ fornece uma tabela de roteamento para aquele nó fonte
    - ↳ Após k iterações, conhece o caminho de menor custo para k destinos

/ /

NOTAÇÃO

$c(i,j)$ → custo do enlace do nó i ao nó j
(∞ se não houver enlace entre i e j)

$D(v)$ → valor atual do custo da fonte até o destino v

$P(v)$ → nó predecessor ao longo do caminho da fonte ao nó v

N → conjunto de nós cujo caminho de menor custo é definitivamente conhecido

INICIALIZAÇÃO

$N' = \{u\}$

para todos os nós v:
  se v é adjacente a u, então:
    $D(v) = c(u,v)$
  senão:
    $D(v) = \infty$

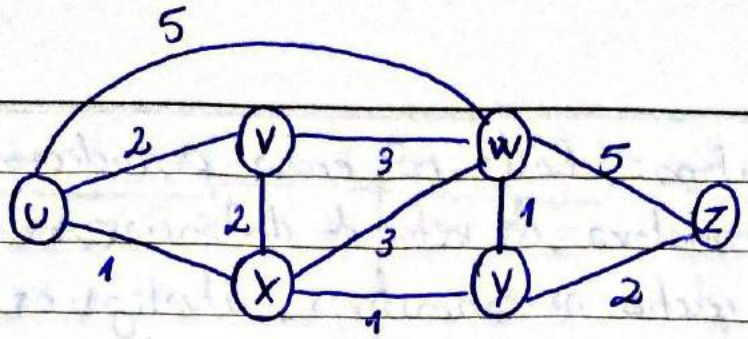
LAÇO

Loop:
Ache w não em N' tal que $D(w)$ é mínimo
Acrescenta w a N'
Atualiga $D(v)$ para todo nó adjacente a w que não está em N'.

$$D(v) = \min\{D(v), D(w) + c(w,v)\}$$

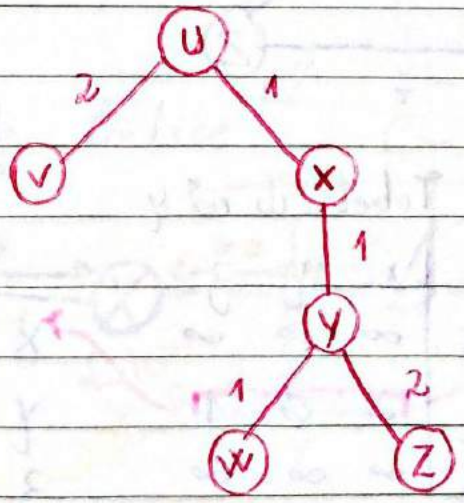
Até que todos os nós estejam em N'

Obs.: Junto com $D(v)$, o $P(v)$ sempre é atualizado.



[...] Também tem z

Passo	N'	D(v), P(v)	D(x), P(x)	D(y), P(y)	D(w), P(w)
∅	u	2, u	1, u	∞	5, u
1	ux	2, u	—	2, x	4, x
2	uxy	2, u	—	—	3, y
3	uxyv	—	—	—	3, y
4	uxyvw	—	—	—	—
5	uxyvwz	—	—	—	—



- Algoritmo de Vetor de Distâncias

  * Equação de Bellman-Ford (fluxo máximo em redes)

$d_x(y)$ = custo do caminho de menor custo de x para y.

$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$ em que o mínimo é calculado para todos os vizinhos v de x.

→ Esses valores são os vetores de distância que dão nome ao algoritmo e que serão enviados entre os vizinhos

/ /

↳ Ideia do algoritmo: Cada nó envia periodicamente sua própria estimativa de vetor de distâncias aos vizinhos. Quando o nó x recebe as estimativas, atualiza os seus $D_x(Y)$ usando a equação de Bellman-Ford.

↳ Cada nó, de forma assíncrona: Espera por mudança no custo do enlace local na mensagem do vizinho; Recalcula Estimativas; Se o DV para qualquer destino muda, notifica os vizinhos.

→ O algoritmo está sempre aqui

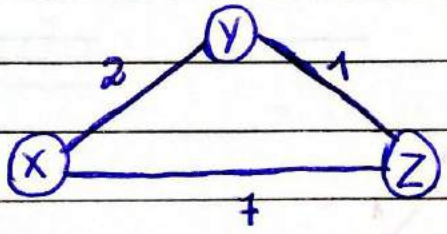


Tabela do nó x

Custo Até	x	y	z
x	∅	2	7
y	∞	∞	∞
z	∞	∞	∞

Tabela do nó y

	x	y	z
x	∞	∞	∞
y	2	∅	1
z	∞	∞	∞

Tabela do nó z

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	∅

	x	y	z
x	∅	2	3
y	2	∅	1
z	7	1	∅

	x	y	z
x	∅	2	7
y	2	∅	1
z	7	1	∅

	x	y	z
x	∅	2	7
y	2	∅	1
z	3	1	∅

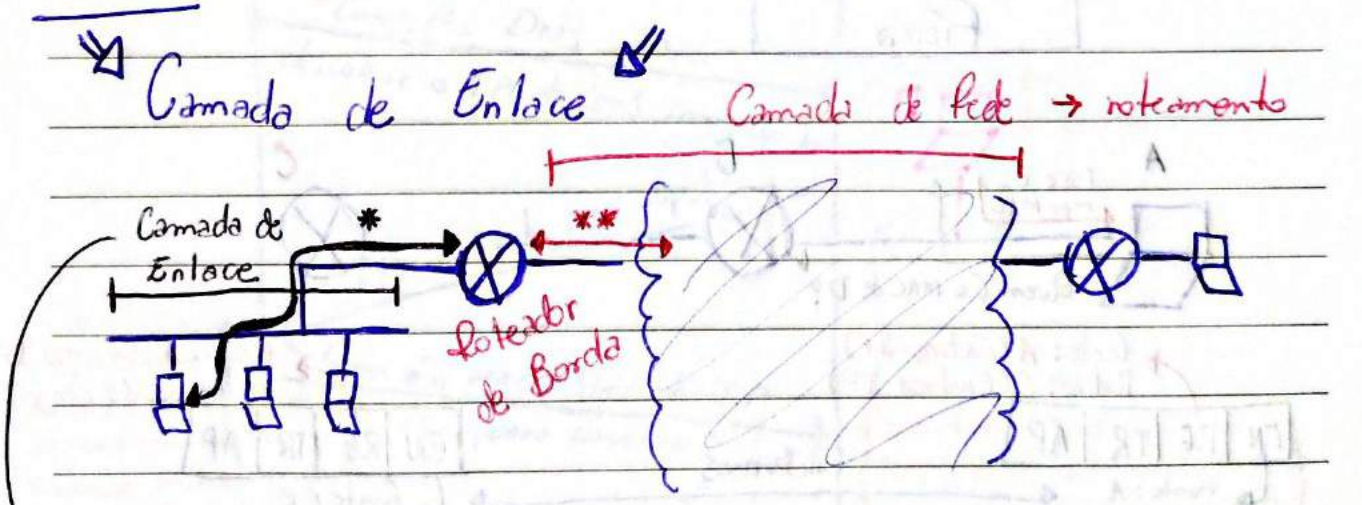
	x	y	z
x	0	2	3
y	2	∅	1
z	3	1	∅

	x	y	z
x	0	2	3
y	2	∅	1
z	3	1	∅

	x	y	z
x	∅	2	3
y	2	∅	1
z	3	1	∅

12/06

Camada de Enlace



→ Objetivo: Transferir datagramas de um nó para o nó adjacente sobre um enlace.

Roteador de Borda

Se * é uma rede sem fio, serão usados protocolos de rede sem fio.

Se ** é uma rede óptica, serão usados protocolos de uma rede óptica.

O fabricante e a menos significativa, a interface.

É administrada pela IEEE

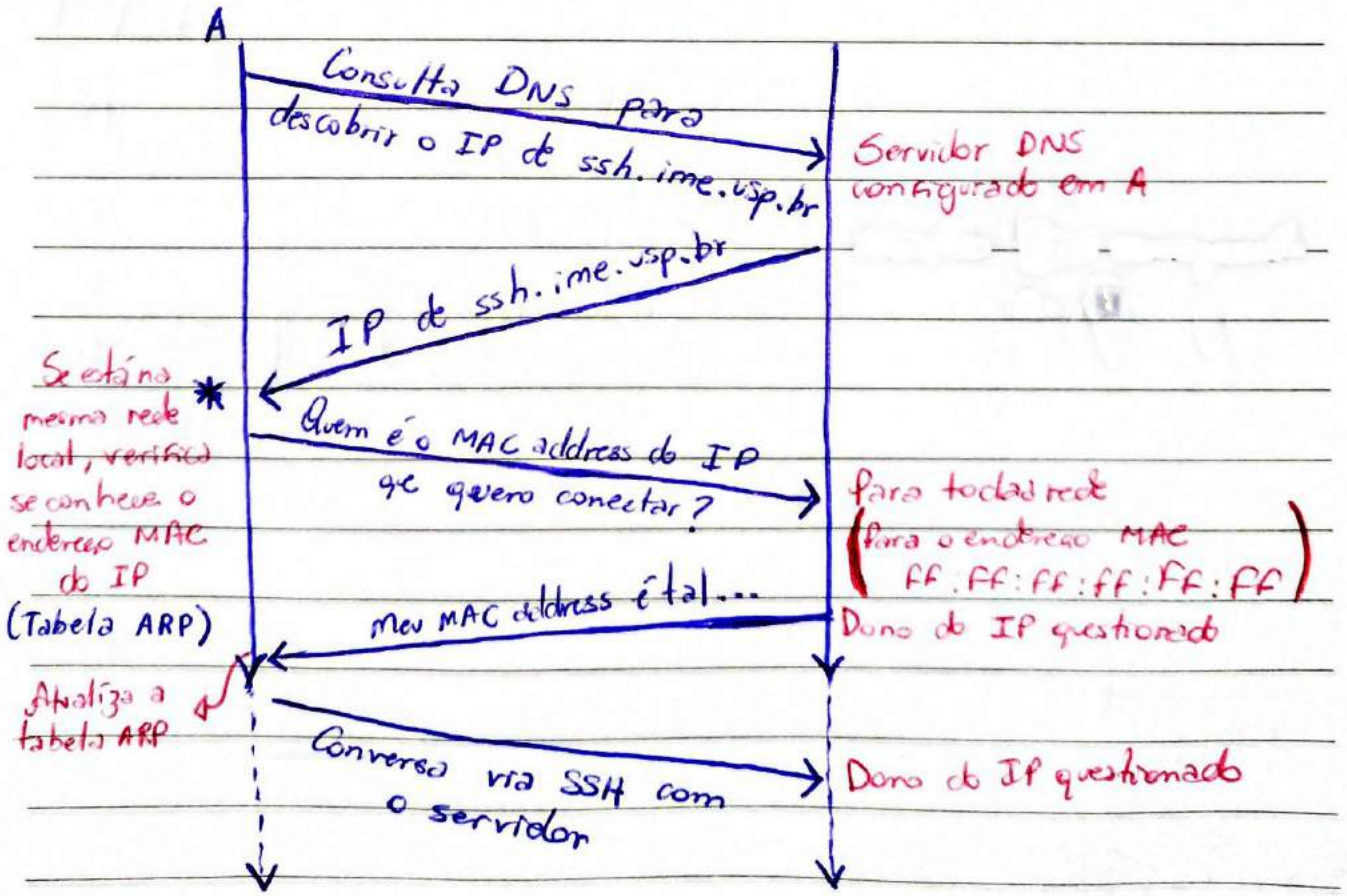
Ex.: 4d:56:13 : ab:7f:4c

4d:56:13 — Fabricante

ab:7f:4c — placa

Protocolo ARP (Address Resolution Protocol)

Objetivo: Fazer a descoberta de endereços MAC em uma rede local.

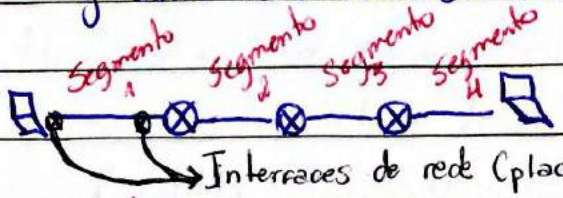


/ /

Objetivos mais específicos:

- controle de fluxo
- detecção de erros
- correção de erros

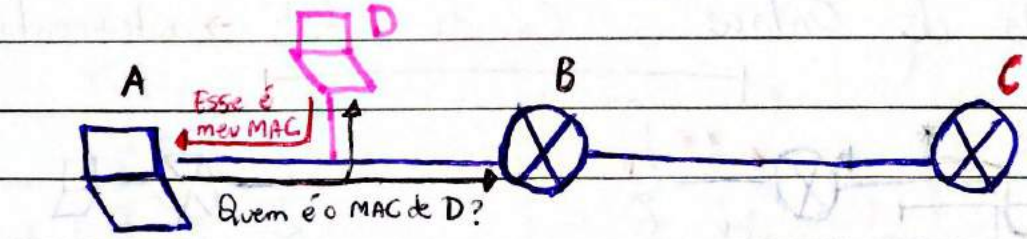
(Endereçamento) garantir os vários sentidos da comunicação (Half e Full Duplex)

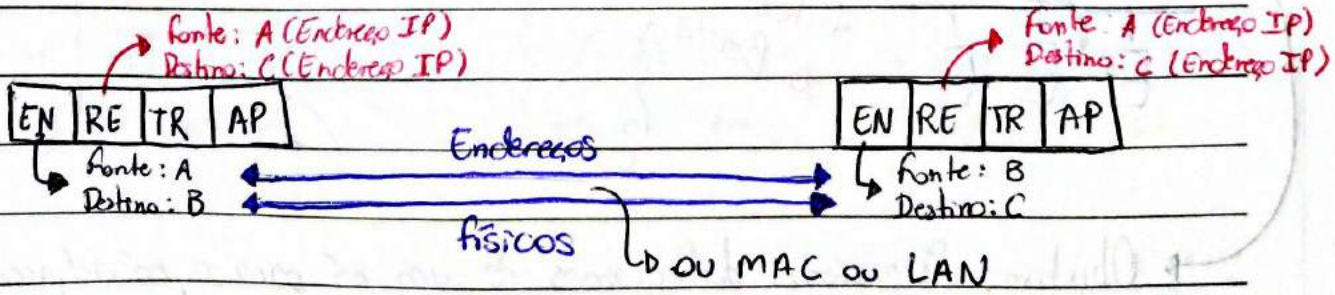


Mantidas à medida que o quadro vai transitando entre segmentos

Aplicação
Transporte
Rede
Enlace
Física

→ Muda a cada segmento





- Endereço MAC identifica unicamente uma placa de rede em um segmento de rede
- Vem de fábrica pré-definido na placa de rede

tilibra Endereço de 48 bits em que a parte mais significativa identifica

/ /

Ex.: Paridade ⇒ Constrói uma matriz e indica paridade de linhas e colunas

Ex.: Checksum ⇒ Dados são tratados como números em blocos de x bits (payload)
Esses x bits são somados, o resultado da soma é invertido (complemento) e anexado no cabeçalho

Detecção e Correção de erros

meio sem fio ≠ óptica