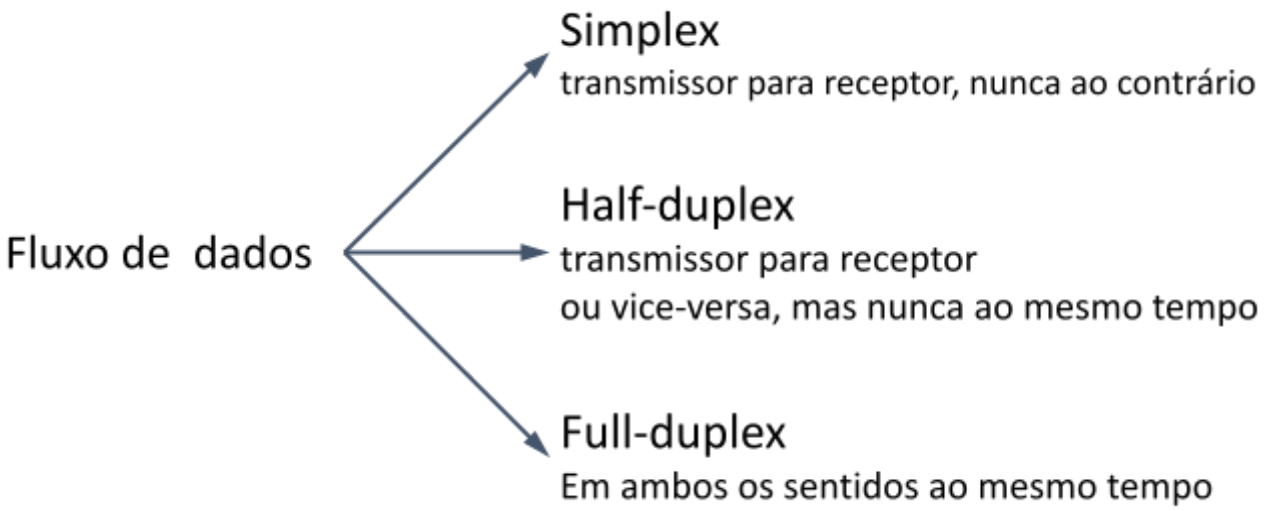
AULA 1 - 6/Agosto/2025

- Apresentação da Disciplina
- Avaliação

Introdução

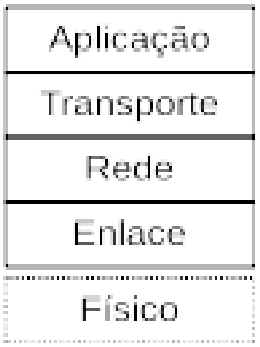
- Comunicação de dados: relação entre
  - mensagem: a informação que se deseja enviar
  - transmissor: quem envia
  - receptor: quem recebe
  - meio de transmissão: caminho físico por onde a msg viaja do transmissor até o receptor.
  - protocolo: regras para que a comunicação ocorra

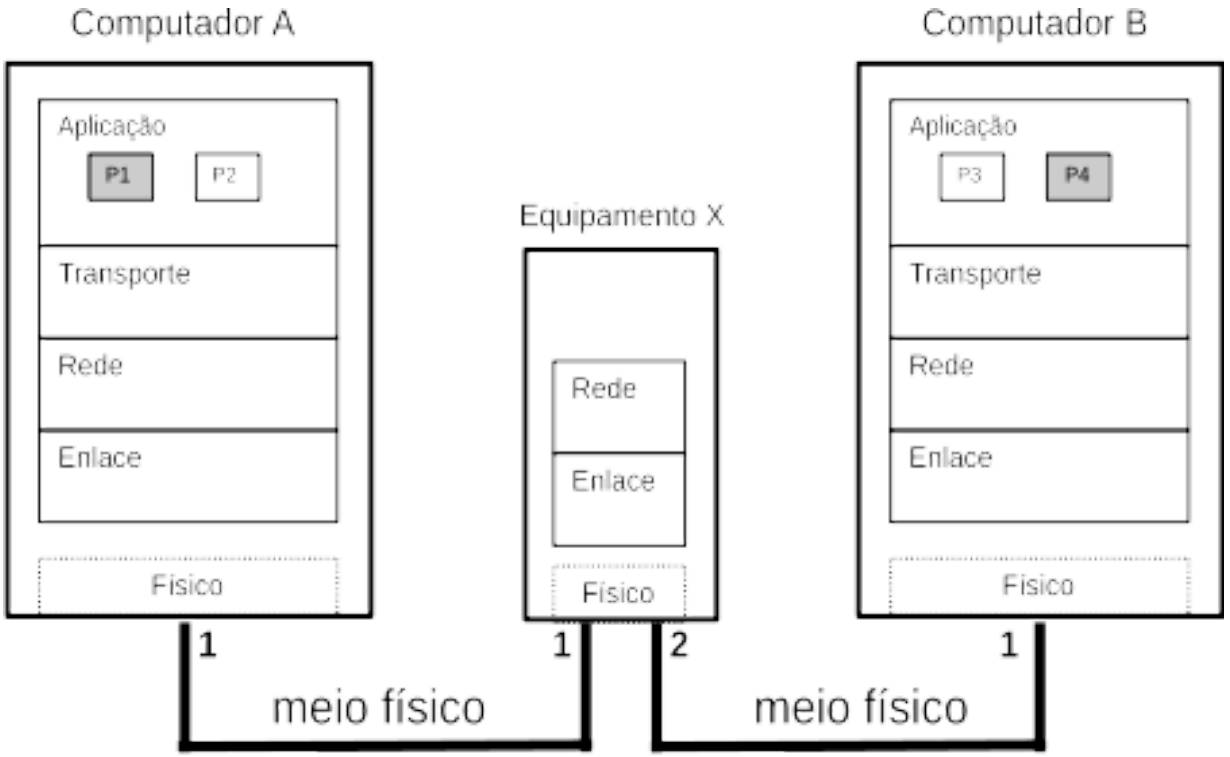


- Direção do fluxo de dados:
  - *simplex*: apenas uma via, mão única. Do transmissor para receptor, por exemplo. (desenhar)
  - *half-duplex*: permite ir ou vir, mas nunca ao mesmo tempo
  - *full-duplex*: enviar e receber ao mesmo tempo
    - ou link possuir dois caminhos
    - ou link é compartilhado enviar/receber (multiplexação)
- Redes, tipo de conexão:
  - ponto a ponto: apenas um receptor e um transmissor, todo o link é reservado para esta transmissão.
  - multiponto: vários hosts
- Topologia física:
  - malha:
    - cada host tem link dedicado a outro host.
    - É ponto a ponto.
  - estrela:
    - cada host se comunica com um centralizador.
    - É ponto a ponto (ex: hub)
  - barramento:
    - um único meio físico compartilhado
    - problema de escalabilidade
  - anel:
    - máquinas ligadas umas às outras em anel.
- Modelos:
  - redes por comutação
    - exemplo telefone, rede ATM
    - redes por pacotes. exemplo: ethernet

Arquitetura de redes

- Modelos de camadas
  - Cada camada com uma função específica
  - cada camada comunica-se com superior e a inferior
  - possui apenas funções de envio/recebimento de dados entre camadas
  - vantagens:
    - troca de camada por outra transparente
    - alteração na lógica de uma camada não interfere nas demais
- Modelo OSI (7 camadas)
  - **Nível físico:** sinais digitais (0v = 0, 5v = 1)
  - **Enlace:** dá interpretação aos bits. Divide em quadros, com seus limites e tamanhos e livre de erros. Uso típico de CRC.
  - **Rede:** como chegar no destino? Roteamento.
  - **Transporte:** dividir dados da camada de sessão em pedaços aceitáveis, garantir a ordem de chegada e a se todos chegaram.
  - **Sessão:** separa usuários/programas de máquinas.
  - **Apresentação:** padronização dos dados. Ex. representação num inteiros.
  - **Aplicação:** protocolos dos programas.
- **Modelo TCP/IP**
  - **Físico:** conversão de bits em sinais elétricos
    - Na verdade não pertence ao modelo TCP/IP
  - **Enlace:** quadros, controle de erros. *Ethernet*, *Frame Relay*, etc
  - **Rede ou Internet:** Responsável pelo roteamento. Ex IPv4, IPv6
  - **Transporte:** separa usuários/programas. Ex: TCP, UDP
  - **Aplicação:** Aplicações de usuários. Ex: HTTP, SMTP, DNS, etc
- cada camada coloca cabeçalhos
  - desenho com cabeçalhos



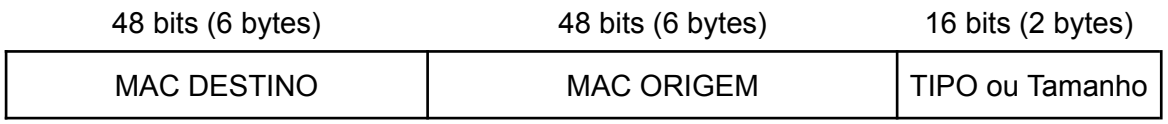


Nível de Enlace

Padrão *Ethernet* (802)

- Formas de conexão:
  - 10Base2: barramento, colisão
  - 10BaseT: estrela, uso de *hub* (concentrador), colisão

- Placas tem *Mac Address*: 48 bits
  - 3 bytes iniciais: fabricante
    - 00AA00: Intel 02608C: 3COM
  - 3 bytes finais: definido pelo fabricante
- Cabeçalho *Ethernet*: PREÂMBULO:SFD **DST:ORI:TAMANHO** (16bits) CRC
  - Preâmbulo: 56 bits alternando 1 e 0
  - SFD (*Start Field Delimiter*): um byte de 0 e 1 alternado. Indica início de quadro
    - Preâmbulo e SFD são adicionados na camada física, logo não fazem parte do quadro
  - DST: (48 bits ou 6 bytes) Endereço MAC da placa de destino
  - ORI: (48 bits ou 6 bytes) Endereço MAC da placa de origem
  - Tamanho: define o tamanho **apenas** da parte dos dados.
- Endereçamento:
  - *unicast:* apenas para um
    - endereço de origem deve ser SEMPRE *unicast*
    - sempre que o byte mais significativo for par é *unicast*
      - Exemplo: 0**2:**05:04:03:2E:4A
  - *multicast*: para vários
    - sempre que o byte mais significativo for ímpar é *multicast*
      - Exemplo: 0**1**:00:5E:XX:XX:XX: Multicast para IPv4 224 (RFC1112)
  - *broadcast*: para todos da rede
    - FF:FF:FF:FF:FF:FF
- Suporta até 1500 octetos **de dados**
  - por razões históricas: 1500 era um valor ideal para minimizar colisões.
  - Ainda hoje: qualquer quadro *Ethernet* não tem mais do que 1500 octetos de dados
  - Obs: existe uma técnica que, mesmo sendo RFC, não é 100% funcional, chamada de *Jumbo Frame*, onde se pode ir além dos 1500 octetos. Para que ela funcione, todos os equipamentos de uma rede, sem exceção, devem suportar *Jumbo Frame*. Por ser uma técnica fora do padrão Ethernet, irei considerar 1500 como o tamanho máximo do Ethernet



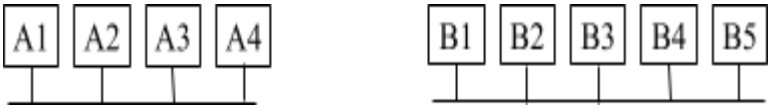
Cabeçalho Ethernet

- Campo *ethernet* TAMANHO:
  - se tamanho <= 1500 (05DC) expressa tamanho
  - se tamanho > 1500, expressa tipo de pacote
  - tipos mais comuns:
    - 0800: Ip versão 4
    - 0806: Arp
    - 8037: IPX
      86DD: Ip versão 6
  - Observação:
    - tamanho de dados MÁXIMO em um quadro *ethernet*: 1500 octetos
    - tamanho de dados MÍNIMO em um quadro *ethernet*: 46 bytes
      - se dados a serem enviados forem < 46 bytes, completa com *padding*
      - dados MENORES do que isto podem fazer com que a máquina NÃO DETECTE uma colisão que existiu (CSMA/CD)

- Nível Enlace só passa para camadas superiores:
  - se *broadcast*
  - se for destino = MEU MAC
  - FF:FF:FF:FF:FF:FF significa Todos. *broadcast*
  - Se estiver participando de algum *multicast* e o pacote for para o MAC do grupo
  - Ou se estiver configurada para atuar em modo promíscuo
- *broadcast* deve atingir TODAS as estações da rede
  - Definição de domínio de *broadcast*
  - Uma rede *ethernet* = domínio de *broadcast*
- Usa CSMA/CD
  - Definição de domínio de colisão

Topologia lógica vs topologia física

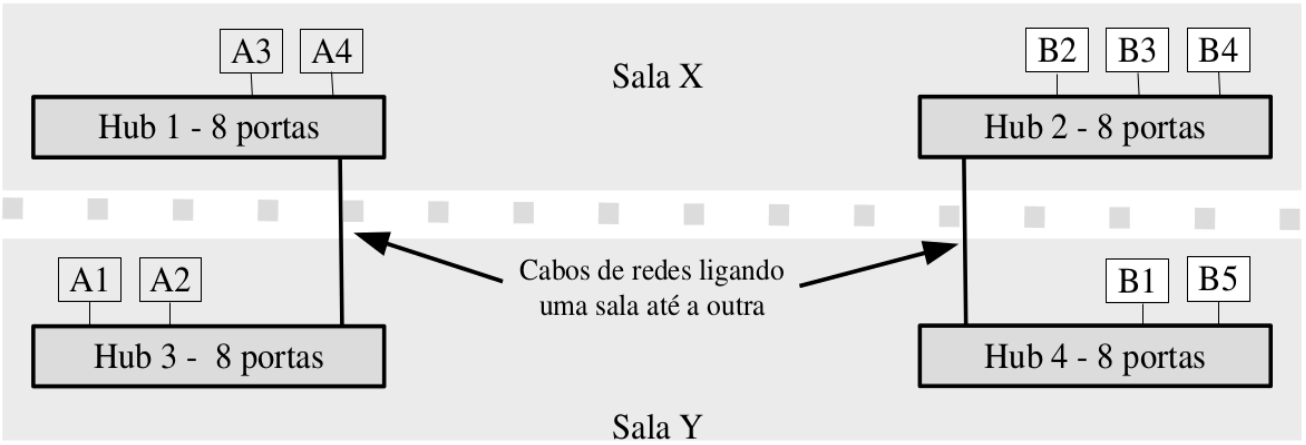
- Um exemplo
  - Rede A: A1, A2, A3 e A4
  - Rede B: B1, B2, B3, B4 e B5



  - Topologia lógica

  - topologia física usando hubs e todas as máquinas em uma mesma sala

  - topologia física usando
  - hubs e com A1, A2, B1 e B5 em outra sala.



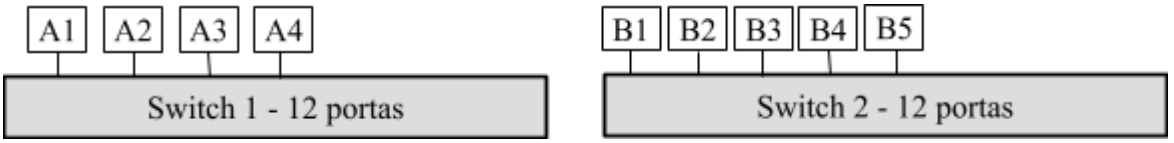
  - Foi necessário ligar o HUB 1 no HUB 3 usando um cabo de rede que sai da sala X até a sala Y. O mesmo ara o HUB 2 e HUB 4
  - Isto se chama cascateamento.
  - não se pode colocar B1 e B5 nas portas sobrando do Hub 3 pois aí estaria no mesmo domínio de *broadcast* da rede A

- Ethernet usando hubs: gera COLISÕES:
  - Se colisões forem muitas, solução: segmentar a rede (Não dividir!!)
  - Necessários equipamentos para passar *broadcast* e quadros ethernet:
    - BRIDGES!!

- BRIDGES:
  - Bridge precisa ter tabelas de MAC ADDRESS
  - Precisa considerar TODOS os quadros (modo promíscuo)
  - Repassar quadros somente:
    - se MAC destino = *broadcast*
    - se MAC destino estiver no outro lado da *bridge*
  - Como o bridge monta esta tabela?
    - Estática: configurado no equipamento
    - dinâmico: aprendizado

Redes locais virtuais (VLAN)

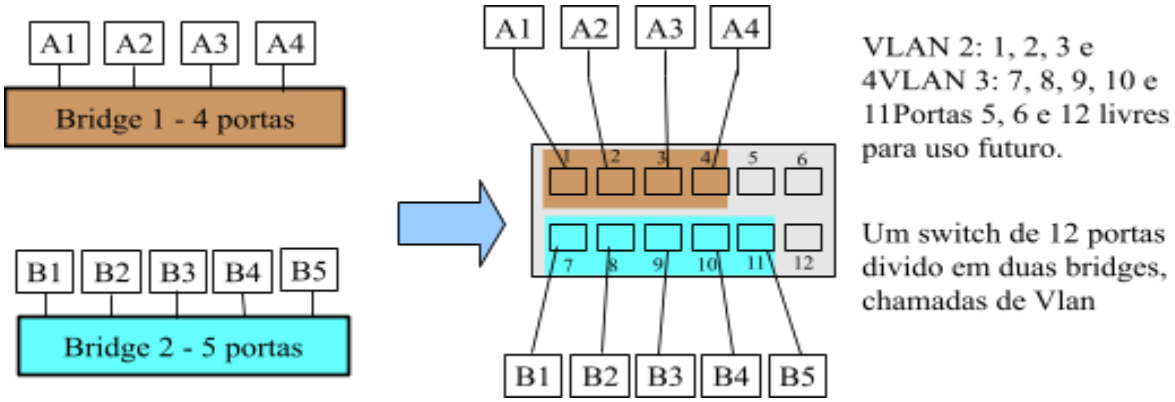
- *Switch*: inicialmente é apenas uma bridge ethernet
  - porém com Muitos lados
  - chamados de portas
  - 12, 24 ou 48 portas (alguns com 48 + 2 Giga)
  - Exemplo: Duas redes, rede A com 4 máquinas e rede B com 5 máquinas
    - Solução 1: adquirir 2 *switches* de 12 portas cada



      - 8 portas livres no switch 1
      - 7 portas livres no switch 2
      - 15 portas desperdiçadas.
    - Solução 2: adquirir apenas um switch gerenciável que permita o uso de vlans

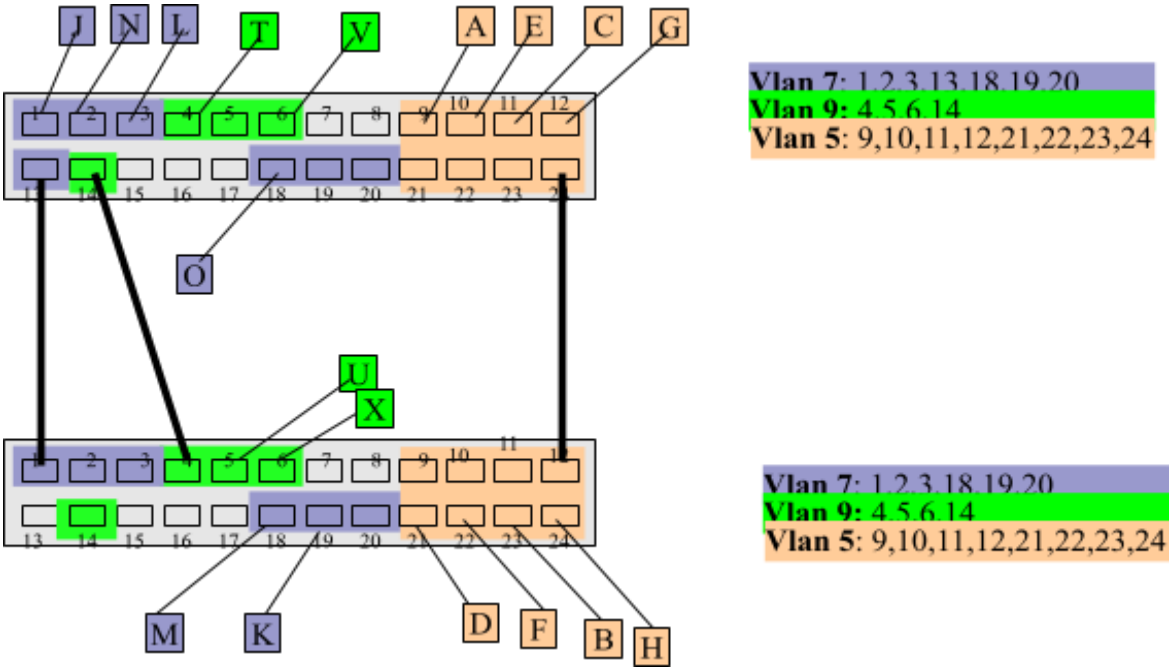
- **VLANs**
  - divisão em dois ou mais domínios de *broadcasts* (Vlan)
    - Ou, se preferir, divisão de um *switch* em várias *bridges*
  - vlans identificadas por números
  - não existe ligação entre uma Vlan e outra
  - cada vlan comporta-se como se fosse uma nova bridge
    - Cada VLAN tem a sua tabela de comutação
  - Um *switch* que nunca foi configurado de 12 portas possui apenas:
    - VLAN 1: todas as portas (ele é uma única *bridge*)
  - Na solução 2, é preciso logar-se neste *switch* e:
    - Criar a VLAN da rede A dando-lhe um número. No caso do desenho, número 2. Também é possível dar um nome, uma string, para facilitar o uso.
    - Criar a VLAN da rede B, no caso, a VLAN 3.
    - Inserir na VLAN 2 as portas 1, 2, 3 e 4
    - Inserir na VLAN 3 as portas 7, 8, 9, 10 e 11

- NÃO ESQUEÇA:
  - Cada VLAN é uma *bridge*.
  - Cada bridge é um domínio de *broadcast*
  - Domínio de *broadcast* == rede ethernet



  - Um domínio de *broadcast* não consegue conversar com outro domínio de *broadcast* (não no nível de ENLACE).

- Problema: pontos de vlans ligados distantes.
- Desenho de 2 andares, um SW por andar e 3 redes
  - Andar 1: A, C, E, G, J, L, N, O, T, V
  - Andar 2: B, D, F, H, K, M, U, V.
  - Solução 1: usar três cabos para ligar os dois *switches*



  - Ligação de uma porta da VLAN 7 do SW1 a uma porta da VLAN 7 do SW2
  - Ligação de uma porta da VLAN 5 do SW1 a uma porta da VLAN 5 do SW2
  - Ligação de uma porta da VLAN 9 do SW1 a uma porta da VLAN 9 do SW2
  - Montar tabelas MACs (três tabelas por SW)

- Protocolo IEEE 802.1q
  - Tipo de quadro: 8100
  - 12 bits para identificar a vlan
    - também conhecidas como vlans *tagged*
    - Switch altera os quadros *(reescreve)*
    - total de 4094 vlans (a vlan 0 e 4095 não podem ser usadas)

3 bits	1 bit	12 bits	16 bits
PCP 802.1p	CFI	VLAN ID (1 a 4094)	Tipo do quadro original

Cabeçalho 8021q

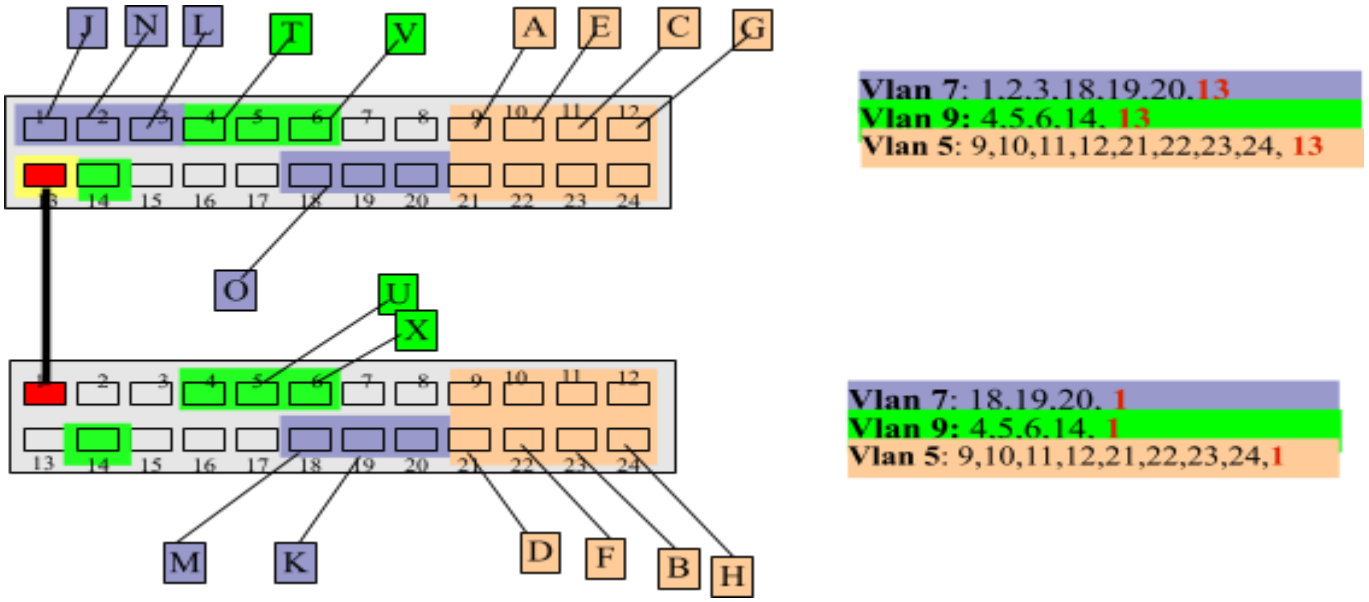
  - Cabeçalho:
    - PCP: 3 bits que determinam a prioridade de acordo com o protocolo 802.1p (serão ignorados nesta disciplina)
    - CFI: um único bit que identifica o formato do mac address. Serve apenas para diferenciar Ethernet de token Ring. Será sempre 0 para Ethernet
    - VLAN ID: o número da Vlan em 12 bits. O valor 0 e o valor 4095 (todos os bits em 0 e todos os bits em 1) não podem ser usados o que deixa uma faixa de 1 até 4094, ou seja, apenas 4094 vlans
    - TIPO do quadro original: o que tinha no campo TIPO do cabeçalho ethernet original (pois agora ele tem 8100)

  - Exemplo prático: Máquina **A** falando com a **D**
    - A gera quadro *ethernet* corretamente
    - SW 1 vê na sua tabela que **D** está em sua porta 13
    - Ao enviar pacote pela porta 13, o SW coloca o tag da vlan **(5)**
    - Ao chegar no SW2 pela porta 1, SW 2 identifica a vlan e retira tag
    - olha em sua tabela (para VLAN 5) onde está **D**
    - acha na porta 21 e repassa quadro original ao **D** (sem o 802.1q)
  - Problema: se uma máquina estiver ligada na porta 13 de SW1, a máquina não reconheceria o protocolo 8100 e descartaria o quadro.
    - Mas como isto é nível de enlace, é só programar suporte
    - Máquina PODE reconhecer (Linux tem suporte há muito tempo
  - Configuração de SWITCHEs
    - Porta: está em uma VLAN ou
    - Configurada para ser 802.1q

- Exemplo de um quadro:
  - Considere que a máquina J (MAC 00:00:01:02:03:04) da vlan 7 está enviando um quadro 0806 (ARP) para *broadcast*.
  - o formato deste pacote é o mostrado no quadro A

DST (broad)	Origem (J)	Tipo (ARP)	
FFFFFFFFFFFF	000001020304	0806	dados dados...

Quadro A



  - Como é *broadcast*, a bridge joga em todas as suas portas (obs: cada vlan é uma bridge)
    - pertencem a bridge onde o J está nas seguintes portas: 1, 2, 3, 18, 19, 20.
    - A porta 13 pertence a vlan 7, mas com TAG (isto é, 802.1Q)
  - O quadro A é jogado, como está nas portas:
    - 2 para chegar ao N
    - 3 para chegar ao L
    - 18 para chegar ao O
    - 19 e 20 pois é da mesma vlan 7
  - O quadro A é também jogado na porta 13 do SW
    - como esta porta é 802.1, o quadro A é alterado como mostrado no quadro B:

DST (broad)	Origem (J)	Tipo	PCP	CFI	VID	TORIG	
FFFFFFFFFFFF	000001020304	8100	???	0	07	0806	dados ...

Quadro B

  - O quadro B (com a tag) chega na porta 1 do segundo switch
    - este lê o cabeçalho 802.1Q e vê que se trata de vlan 7
    - identifica quais portas suas estão na vlan 7:
      - 18, 19 e 20 (além da porta 1 em 802.1q)
  - Switch 2 remove o cabeçalho 802.1q restaurando o quadro original (quadro A)
  - joga o quadro A nas portas 18, 19 e 20
    - Quadro chega em todas as portas que são da vlan 7
      - e chega em todos sem o cabeçalho 802.1q, apenas o quadro original (quadro A)

        - Este quadro alterado (quadro B, 802.1q) deve ficar internamente de switch a switch. Mesmo assim o switch altera o quadro de maneira correta, inclusive ajustando o CRC existente no quadro
            - se o quadro B for entregue para uma máquina (acidente?) Ela verá um quadro normal, legítimo, bem formado.
            - Pode apenas ignorar por não conhecer o protocolo 8100 (802.1q)

Revisão sobre redes:

- Uma máquina só conversa com uma máquina de seu mesmo domínio de *broadcast*
- E se quer falar com uma máquina fora?
    - Necessário repassar para alguém
    - Função do *gateway*
- Ideia do roteamento
    - Passagem do quadro *ethernet* para o *gateway*
    - Este roteamento é realizado pelo nível de Rede

Nível de Rede (Internet)

- Responsável pelo Roteamento
- Como chegar ao destino?
- Se mesmo domínio de *broadcast*:
    - destino é atingível, basta colar MAC destino no quadro
        - E obtém-se o MAC por ARP
- se destino não for mesmo domínio de *broadcast*
    - necessário repassar quadro para o *gateway*
    - quadro vai para o MAC ADDRESS do gateway
        - necessário ARP para obter o MAC do gateway
- Endereço de rede para saber se é ou não mesmo domínio de *broadcast*
    - Ipv4 ou IPv6
- Protocolo ARP:
    - “Quem é o Ip 192.168.0.10”
    - “Eu sou o IP 192.168.0.10 e meu endereço é 20:30:40:50:60:70”
    - Tabela Cache de ARP:
        - associações de ARP = IP
            - estático
                - inserido pelo administrador
                - máquina não realiza ARP para as entradas estáticas
                - útil por questões de segurança
                - Exemplo Linux:

                  arp -s 192.168.0.15 20:30:40:50:60:70
            - dinâmico
                - realiza ARP sempre que precisar
                - otimização: manter associações na tabela por algum tempo

IPv4

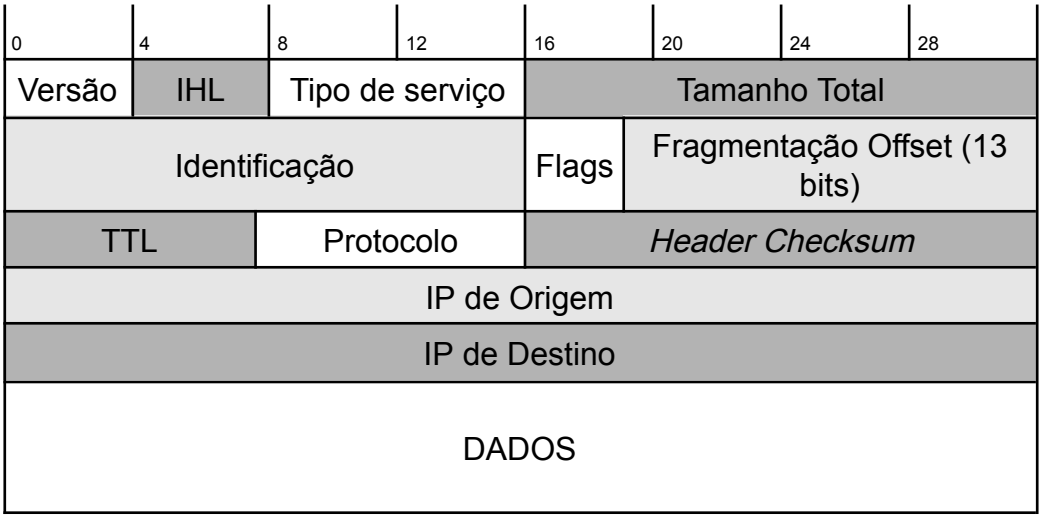
- número IP representado por 32 bits, um "*unsigned long int*" do C
- Exemplo:

```
11000000 10101000 00000000 00001010
   C0       A8       00       0A
3232235530 (decimal)
```

- Números binários e Hexa são de difícil compreensão para nós.
- uso em forma Decimal

```
11000000 10101000 00000000 00001010
  192       168       0       10
```

  - usando pontos: 192.168.0.10
- Cabeçalho IPv4



- cabeçalho de um datagrama IP (5 words mínimas)
  - Versão: 4bits (0000 a 1111). IPv4=0100
  - IHL: 4 bits. Tamanho do cabeçalho em words (5 words mínimo)
  - Tipo de serviço 8bits:
    - Procedência (3 bits: 0-> normal, 7-> controle de rede)
    - *Delay* (3 bits)
    - outros: Ignorado
  - Tamanho do datagrama (em bytes, dados + cabeçalho): 16 bits. até 64 Kbytes
  - Identificação: 16 bits: quando fragmentado identifica um mesmo datagrama.
  - Flags: 3 bits (nãoUSADO:DF:MF - *Don't Fragment*, *More Fragments*)
    - DF e MF para Controle de Fragmentação de datagramas IP
    - DF (Don't Fragment):
      - 1 = Não deve ser fragmentado
      - 0 = Pode ser fragmentado
    - MF (More Fragments):
      - 1 = Tem outros fragmentos
      - 0 = Não tem mais fragmentos (este é o último ou único)
    - se não for possível enviar datagrama sem fragmentar e DF=1, ele é descartado
  - *Fragmentation Offset*: 13 bits (0...8191).
    - Fragmentos calculados em unidades de 8 bytes
      - FO = 00 (este fragmento se encaixa no byte 0)
      - FO = 03 (este fragmento se encaixa no byte 24)
      - FO = 13 (este fragmento se encaixa no byte 104)

        - 8K fragmentos de 8 bytes = 64KiB (tamanho máximo de um Pacote IP)
        - O tamanho mínimo de dados de um fragmento é 8 bytes
        - O tamanho máximo de um cabeçalho IP é 60 bytes (o mínimo é 20)
        - Desta forma, a RFC determina que TODAS as máquinas devem ser capazes de lidar com datagramas de pelo menos 68 bytes (senão não tem como haver a comunicação).
      - Cada fragmento tem o tamanho deste fragmento no campo tamanho
    - TTL: *Time to Live*: 8 bits
  - Protocolo: 8bits
    - ICMP: 0x01
    - TCP: 0x06
    - UDP: 0x11 (17 em decimal)
  - *Checksum*: 16 bits
  - IP origem:
  - IP destino:

Roteamento no IPv4

- Problema: como rotear
  - De alguma forma a máquina sabe que destino não é local e repassa ao gateway
  - através da padronização dos números Ips alocados

- Classes de Ip (não mais usada)
  - Ip dividido em duas partes: Rede e máquina
  - Na parte de máquina, sempre:
    - todos os bits de host em ZERO determina o número da rede
    - todos os bits de host em UM determina *broadcast* de IP

  - Classe A: (**0**RRRRRRR HHHHHHHH HHHHHHHH HHHHHHHH)
    - Primeiro bit em ZERO
    - outros 7 bits identificam a REDE
    - Total de 128 redes (rede 00000000 a 01111111)
    - Ips: 0.x.x.x até 127.x.x.x são classe A
    - 24 bits identificam máquinas (total de 16.777.216 - 2 máquinas)
    - Exemplo: Classe A 5.x.x.x
      - Endereço de rede: 5.0.0.0
      - Endereço de *broadcast*: 5.255.255.255

  - Classe B: (**10**RRRRRR RRRRRRRR HHHHHHHH HHHHHHHH)
    - Primeiros 2 bits 10 identificam classe
    - Outros 14 bits identifica a rede
      - total de 16384 redes
      - rede de 128.x.x.x até 191.x.x.x
    - 16 bits máquinas (65536 - 2 máquinas em cada rede)

  - Classe C: **110**RRRRR RRRRRRRR RRRRRRRR HHHHHHHH)
    - Primeiros 3 bits 110 id classe
    - Outros 21 bits o número da rede (2097152 redes)
    - de 192.x.x.x até 223.x.x.x
    - 8 bits para máquinas (256 máquinas - 2 = 254 hosts)

  - Classe D: (**1110**xxx)
    - Reservado para multicast

        - não usado para máquinas na Internet
        - de 224.x.x.x até 239.x.x.x

    - Classe E: (**1111**xxx)
        - Reservado, sem uso
        - de 240.x.x.x até 255.x.x.x
        - 255.255.255.255: endereço de *broadcast* global

- Uso de classes facilita roteamento
    - diminui os custos
    - não necessita tabelas muito grandes
- em pouco tempo:
    - todos classe B usados
    - classe A muito grande e C muito pequeno para maioria das empresas
    - Apenas classe C disponíveis
    - necessário redividir Ips

- Classe CIDR (***C**lassless **I**nter-**D**omain **R**outing*)
    - Introduzido em 1993
    - Não fixar o número de bits para rede.
    - Qualquer ip, não importa de qual classe seria, pode ter quantos bits quiser para rede.
- Máscara de rede
    - IP + Máscara
    - Roteamento por tabelas
    - Cada equipamento deve saber:
        - Seu IP
        - Sua rede (pela máscara)
    - Rede expressa em máscara: números de 1's usados
    - Exemplo:
        - Ip 192.168.10.10 pertence a uma rede de número
        - 192.168.10.0 (como um classe C).
        - Dos 32 bits, 24 designam a rede e 8 a host

```
IP: 11000000 10101000 00001010  00001010
    [--------   Rede  ------>]  [<Host>]
```

    - Dos bits do IP, "diz-se" quais designam a rede sinalizando com 1's

```
IP:        11000000 10101000 00001010 00001010
BITS REDE: 11111111 11111111 11111111 00000000 (255.255.255.0)
```

- Por que uso de máscara?
  - Facilita operações.
  - Exemplo:

    Qual rede pertence o IP 192.168.10.50/24 (isto e, 24 bits iniciais designam a rede) ?

```
  IP: 11000000 10101000 00001010 00110010 (192.168.10.50)
REDE: 11111111 11111111 11111111 00000000 (255.255.255.0)
AND:  11000000 10101000 00001010 00000000 (192.168.10.0)
```

    Outro exemplo: Ip 172.10.20.40/16 e 172.10.150.30/16

```
172.10.20.40 = 10101100 00001010 00010100 00101000
255.255.0.0  = 11111111 11111111 00000000 00000000
        AND  = 10101100 00001010 00000000 00000000 (172.10.0.0)

172.10.150.30= 10101100 00001010 10010110 00011110
255.255.0.0  = 11111111 11111111 00000000 00000000
         AND = 10101100 00001010 00000000 00000000 (172.10.0.0)
```

  - Conclusão: ambos na mesma rede!!
  - Máscara também usada para calcular o *broadcast*
    - uso dos bits de rede INVERTIDOS:

```
MASCARA  = 11111111 11111111 00000000 00000000 (255.255.0.0)
NUMERO IP= 10101100 00001010 10010110 00011110 (172.10.150.30)
MÁSC INV = 00000000 00000000 11111111 11111111 (0.0.255.255)
     OR  = 10101100 00001010 11111111 11111111 (172.10.255.255)
```

    - Para se calcular o *broadcast*, faz-e um OR bit a bit do IP com a máscara invertida (complementada)
    - Internamente os programas manipulam IPs e máscaras como um número INTEIRO de 32 bits
    - Logo, em C, as operações matemáticas para obter rede e *broadcast* são:

```
rede      = ip & mascara;
broadcast = ip | (~mascara);
```

  - Opcionalmente (mais comum) expressa-se a máscara pela notação decimal que ela representa:

```
/24 = 255.255.255.0
/16 = 255.255.0.0
/25 = 255.255.255.128
/23 = 255.255.254.0
```

- Divulgação da página para cálculo de máscara: https://elgio.prof.nom.br/redes/

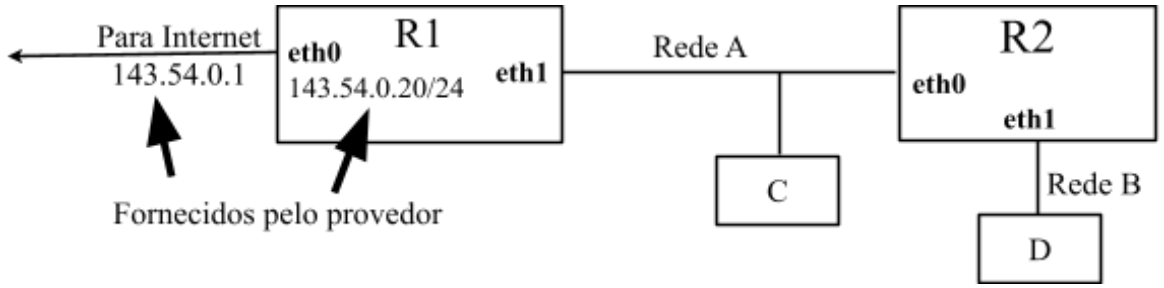
- Ips reservados
  - Para intranet:
    10.0.0.0/8
    172.16.0.0/12
    192.168.0.0/16
    - São chamados de ips **privados**.
    - NÃO conseguem acessar a Internet! (somente com técnicas que estudaremos adiante)
    - Não são "roteáveis" na Internet
      - Não é correto chamar estes ips de "inválidos" ou "não roteáveis". São ips "válidos", pois posso atribuir e são sim "roteáveis" na minha Intranet. O termo correto é IPS PRIVADOS!!
  - Inválidos (reservado):
    - 127.0.0.0/8: reservado para uso local
    - 0.0.0.0/8: reservado
  - Convenção usada na Disciplina:
    - Ips públicos: usados o classe B da UFRGS: 143.54.0.0/16
    - Ips privados: qualquer um, mas eles NÃO TÊM ACESSO a Internet
- Divisão de IPs em várias sub redes
  - Exemplo prático: Considerando um 10.1.0.0/24
    - Se usar inteiro:
      - rede 10.1.0.0 *broadcast* 10.1.0.255
      - total de 254 máquinas na rede (de 10.1.0.1 até 10.1.0.254)
    - ou pode ser dividido em duas /25
      - 10.1.0.0/25 rede 10.1.0.0 *broadcast* 10.1.0.127 (10.1.0.1 até 10.1.0.126)
      - 10.1.0.128/25 rede 10.1.0.128 *broadcast* 10.1.0.255 (10.1.0.129 até 10.1.0.254)
      - Cada /25 ainda pode ser dividido em dois /26
        - 10.1.0.0/26 rede 10.1.0.0 *broadcast* 10.1.0.63 (10.1.0.1 até 10.1.0.62)
        - 10.1.0.64/26 rede 10.1.0.64 *broadcast* 10.1.0.127 (10.1.0.65 até 10.1.0.126)
        - Cada /26 ainda pode ser dividido em 2x /27
          - 10.1.0.64/27 rede 10.1.0.64 *broadcast* 10.1.0.91
            - 30 máquinas de 10.1.0.65 até 10.1.0.90
          - 10.1.0.96/27 rede 10.1.0.96 *broadcast* 10.1.0.127
            - 30 máquinas de 10.1.0.97 até 10.1.0.126
- Dicas:
  - Rede sempre par, *broadcast* sempre ímpar
  - em todo o domínio de *broadcast*, todos devem ter a mesma máscara
  - em todo o domínio de *broadcast*, todos devem calcular o mesmo número de rede e *broadcast*
  - Não pode haver conflitos ou interseções de redes
  - Cada rede precisa ter o octeto múltiplo do tamanho:
    - EX: /26 => rede com 64 ips só pode ser
      - X.X.X.0
      - X.X.X.64
      - X.X.X.128

- X.X.X.192

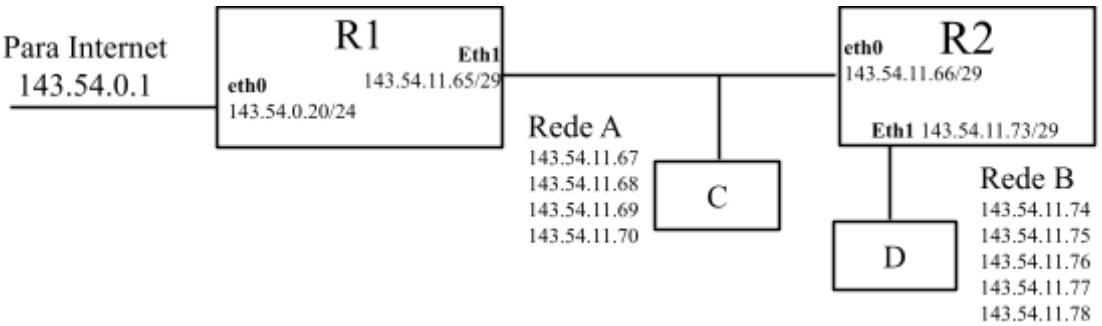
Realização do Trabalho TR5 EM AULA

Roteamento Estático

- Roteamento IPv4
  - Funcionamento do Gateway (exemplo)
  - Roteador: é um software. Toda a máquina é um roteador
  - Rotas:
    - Rotas locais
    - Rotas para cada rede
    - Rota *default*
- Exemplo de uma rede com ips públicos:
  - Provedor te vendeu a faixa 143.54.11.64/28
    - Que você usa como quiser
  - Provedor disse que deves usar o IP 143.54.0.20/24 na interface com ele
    - 143.54.0.0/24 não são ips que você possa usar, são a rede que você tem com o provedor. Isso é bem diferente em uma ADSL
  - Provedor disse que deves usar como gateway 143.54.0.1



  - Situação:
    - Rede A tem 4 máquinas e dois gateways (R1-eth1 e R2-eth0)
    - Rede B tem 3 máquinas e um gateway (R2-eth1)
  - Solução:
    - Dividir o /28 recebido em 2x /29, um para rede A e outro para rede B
    - 143.54.11.64/29 para a rede A (broadcast 143.54.11.71)
    - 143.54.11.72/29 para a rede B (broadcast 143.54.11.79)



  - Rede A não pode ter nenhuma a mais: fechou certo
  - Rede B, que tem 3 máquinas, ainda poderia ter mais 2 máquinas
- Observe que para o provedor não importa como eu dividi e se dividi a minha faixa de Ips públicos: ele vai rotear TUDO que for 143.54.11.64/28 para 143.54.0.20

Tabelas de roteamento:

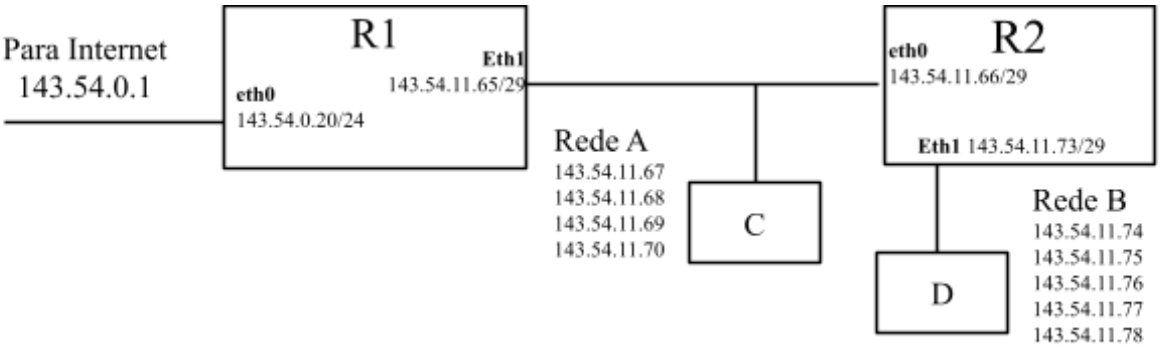


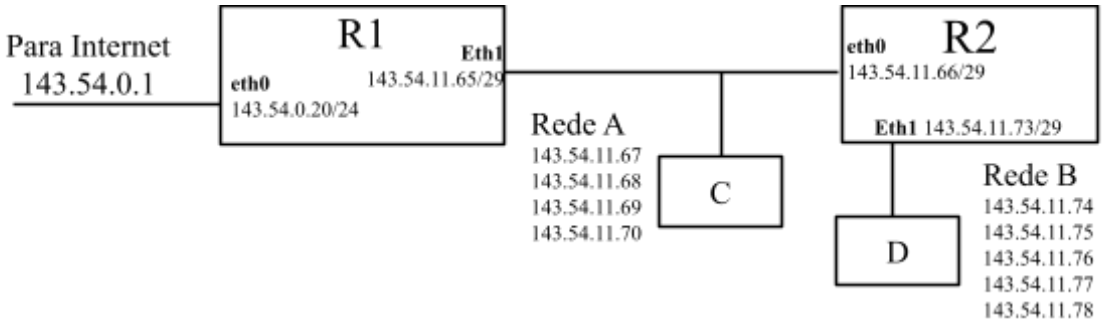
Tabela Roteamento do R2		
Rede	**destino**	**Observação**
143.54.11.72/29	Local eth1	Indica que uma máquina com este IP está na rede local da interface eth1
143.54.11.64/29	Local eth0	Se ele precisar conversar com a máquina C, deve saber que ela está localmente na sua interface eth0
0.0.0.0/0 ou Default	143.54.11.65	Tudo que não casar com as regras anteriores será repassado para 143.54.11.65 (isto é, para o MAC Address dele). Ou seja, se não é Rede A e nem Rede B, então manda para o 143.54.11.65 que ele deverá saber o que fazer.

Tabela Roteamento do R1		
Rede	**destino**	**Observação**
143.54.0.0/24	Local eth0	Não sei o que tem nessa rede. Só sei que minha eth0 faz parte e que é nela que acho o meu gateway (143.54.0.1) informado pelo provedor. Se algum ia eu precisar conversar com a 143.54.0.10 devo saber que ela está nesta rede local da minha interface eth0
143.54.11.64/29	Local eth1	
143.54.11.72/29	143.54.11.66	As máquinas da rede B não estão alcançáveis por este roteador. Para chegar nelas é necessário repassar o pacote para o R2 para o MAC addrress de R2-eth0. Para obter este MAC precisa-se do IP 143.54.11.66 a fim de fazer ARP
Default	143.54.0.1	Tudo que não casar com as regras anteriores será repassado para 143.54.0.1 (isto é, para o MAC Address dele).

- desconhecemos a tabela de roteamento do 143.54.0.1 (do provedor), mas UMA LINHA dela deve ser assim:

Tabela Roteamento do 143.54.0.1		
Rede	**destino**	**Observação**
...	...	...
143.54.11.64/28	143.54.0.20	Faixa de IPS que recebi e que redividi em 2x /29. Mas o provedor de acesso não sabe desta divisão. E nem precisa saber. Só precisa repassar ao meu roteador tudo que for da minha faixa.
...	...	...

- problema com máscara errada
  - a) máquina A e B no mesmo domínio, mas A tem máscara /24 (certa) e B tem /25
    - B usará o *gateway* para se comunicar (não precisava)
  - b) máquina A e X em domínios diferentes, A tem /24 e X tem /25
    - A não falará com X, pois pensará ser da mesma rede
    - X falará com A, mas a resposta morrerá (mesmo motivo)



- problema rota errada ou sem rota
  - Se R1 não tem rota para Rede B?

Tabela Roteamento do R1	
Rede	**destino**
143.54.0.0/24	Local eth0
143.54.11.64/29	Local eth1
Default	143.54.0.1

- Ao enviar para 143.54.11.76 irá bater na tabela
  - Não casa com a primeira regra
  - Não casa com a segunda regra
- Enviará para 143.54.0.1 (default)
- 143.54.0.1 irá usar a sua tabela e irá devolver para R1
  - R1 devolve para 143.54.0.1 que devolve que devolve
  - LOOPING até TTL expirar

  - Diferença entre PLACA e interfaces
    - Quem recebe IP é a interface.
    - Uso comum: cada placa é apenas uma interface.
    - com 802.1q: uma placa e várias interfaces
- **NAT - *Network Address Translation* (RFC 1631)**
  - Associa um IP público a um IP privado
  - Nat estático
    - Tabelas fixas
    - Uso: Ips internos reservados e apenas alguns deles com IP público
    - Tradução: Altera APENAS o cabeçalho IP, trocando:
      - Pacotes que saem: IpPrivadoOrigem = IpPublicoOrigem
      - Pacotes que entram: IpPublicoDestino = IpPrivadoDestino
  - Nat estático precisa alterar dados do cabeçalho IP (Ip Origem ou Destino)
  - Precisa recalcular o *checksum* do cabeçalho IP!!

Aula 8 - 24/Setembro/2025

Exercício EXTRA de revisão.

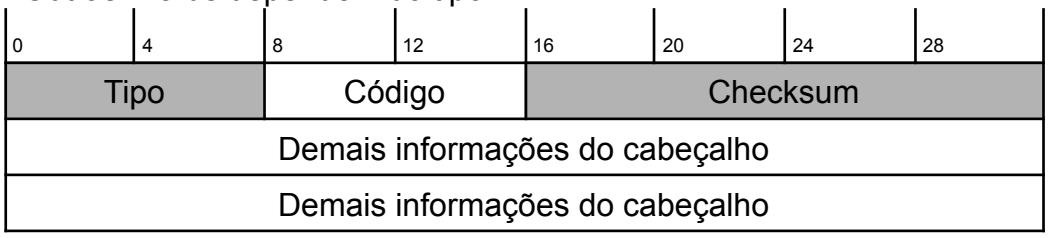
Aula 9 - 1/Outubro/2025

Prova G1

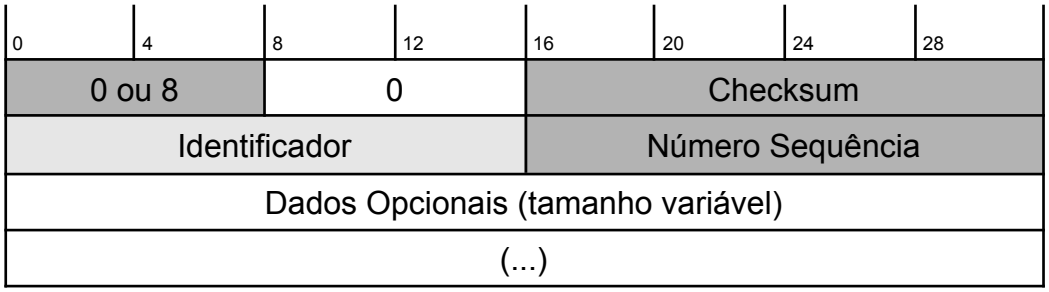
Aula 10 - 08/Outubro/2025

PARTICIPAÇÃO NA SAPIENS

- Pacotes ICMP (*Internet Control Message Protocol*)
  - Retorno de erros
  - protocolo 0x01 no cabeçalho IPv4
  - Cabeçalho (3 ou mais WORDS)
  - Primeira WORD:
    - Tipo: 8 bits
    - Código: 8 bits
    - CRC: 16 bits
  - Outras *Words* dependem do tipo



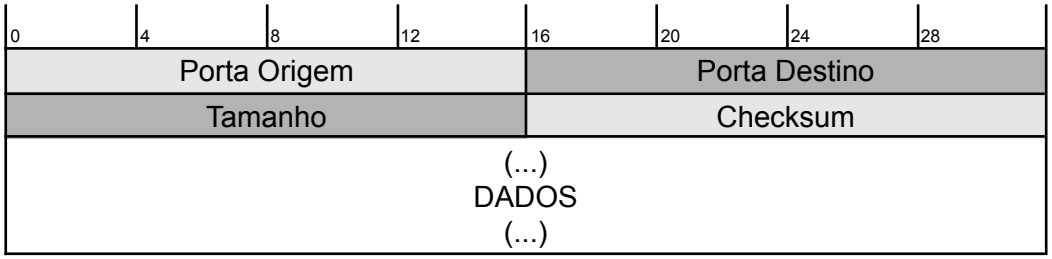
  - Tipos:
    - 0 = Resposta de echo (resposta ao ping)
    - 3 = Destino não acessível
    - 5 = *Redirect* (informa nova rota. PERIGOSO para a segurança!)
    - 8 = Solicitação de echo (ping)
    - 11 = TTL excedeu
    - 17 = Solicitação de máscara
    - 18 = resposta de máscara
  - Código depende do tipo
    - Se tipo for 0 ou 8 (echo) código é 0



    - Se tipo for 3 (destino inacessível):
      - Código:
        - 0 = Rede inacessível
        - 1 = Host inacessível
        - 2 = Protocolo inacessível
        - 3 = Porta inacessível
        - 4 = Fragmentação Necessária, mas pacote tinha DF=1
        - 6 = Rede destino desconhecida
        - 7 = Host destino desconhecido
  - ICMP possui outros serviços:
    - indicação de roteador congestionado
    - sincronização de relógios
    - obtenção da máscara de um host
    - dentre outros

Nível de transporte

- considerações sobre portas e processos
- porta de origem/porta de destino
- Porta modo servidor
- Quando um processo requisita uma porta para que outros se conectem nela
  - Precisa informar ao Sistema Operacional qual porta deseja
- Porta modo cliente
  - Quando um processo inicia a comunicação com o destino
  - Precisa apenas de uma porta de origem para receber os dados de retorno
- Alocação de portas
  - 0 a 1023: portas BEM conhecidas, gerenciadas pela IANA
    - 80 para HTTP
    - 53 para DNS
    - 25 para SMTP
    - 443 para HTTPS
    - 587 para SUBMIT (envio email criptografado)
  - 1024 a 49151: portas registradas. Podem ter serviços
    - Ex:
      - 3306 mysql
    - 4000 geralmente usadas pelos programas de mensagem instantânea
  - Maiores que 49151: efêmeras. Não tem nenhum uso específico registrado.
  - Ver IANA *http://www.iana.org/assignments/port-numbers*
- Sistema Operacional mantém tabela de qual processo é dono de qual porta
  - Desenhar
- Pacote UDP
  - Serviço não confiável
  - Sem garantia alguma
    - não garante recebimento
    - não garante ordem de chegada



    - Cabeçalho:
      - Porta de Origem: 16 bits
      - Porta Destino: 16 bits
      - Tamanho: 16 bits (até 64Kbytes)
      - *CheckSum*: 16 bits

- Aplicação sob UDP:
  - Aplicações "pergunta/resposta"
    - Impossível chegar fora de ordem
    - Pode haver apenas perda do pacote
    - E se houver perda, é só enviar novamente até que uma resposta seja fornecida.
    - Ex: DNS (requisição DNS é curta, cabe em um único pacote UDP)
  - Voip e multimídia "real time"
    - O que chega deve ser reproduzido no destino (pois é áudio ou vídeo ao vivo)
    - Se chegar algum pacote atrasado, já passou o tempo de reproduzir e ele será descartado

        - Se faltar algum pacote, não haverá tempo para reenviar pois ele precisa ser reproduzido AGORA
        - Nestes casos algoritmos trabalham forte na reconstrução de pacotes perdidos, nunca no seu reenvio.
    - Em rede local:
        - perdas são feitas por roteadores congestionados ou rotas falhas
        - desordenamentos são realizados por rotas redundantes ou priorização de pacotes
        - duplicidade de pacotes também são criados por roteadores
        - Como na rede local os pacotes não seguem "rotas" mas apenas de origem para destino direto, o único equipamento que pode introduzir este erros seria o switch
            - não vai haver desordenamento
            - pouco provável que haja perdas
        - Dá para se dizer que uma rede local é "confiável" pela confiabilidade de seus equipamentos, logo é menos provável que existam problemas e o UDP acaba sendo uma boa solução.
    - Multicast e Broadcast
        - Somente em UDP é possível realizar comunicação multicast ou broadcast
        - Exemplo: ghost

Aula 12 - 20/Out/2025

Revisão transporte

- Portas
    - porta de servidor
    - porta de cliente
- UDP

Sugestão de protocolos para introduzir confiabilidade em UDP:

- destino: receberá os dados
- origem: enviará os dados
- **Algoritmo 1 (by Elgio):**
    - Pacotes são numerados de 1 a N, sendo que não se conhece antecipadamente o N
        - discutir porque esta característica é desejável
    - Último pacote tem "-", indicando ser o último
    - destino recebe pacote com dados e imediatamente confirma
    - Formato dos pacotes:
        - **Envio de dados:** "ID,+/-,DADOS"
            - Exemplo:

```
"1,+,Teste "
"2,+,123 "
"3,-,CACA"
Enviado "Teste 123 CACA"
```

- **Formato da confirmação:** "ID,="
    - Exemplo

      "1,="
      "2,="
      "3,="
- origem só envia próximo pacote ao receber a confirmação do anterior.
    - espera um tempo X pela confirmação
    - reenvia se X estourar

- Ao receber o primeiro pacote de um origem, o destino NEGARÁ pacotes de qualquer outro, até que receba o último.
    - Quando destino NEGAR um origem, irá devolver "0,="
    - Com isto é tarefa do origem saber que deve tentar mais tarde, pois o destino está ocupado
    - destino usa IP e porta de origem para identificar origem

- **AVALIAÇÃO ALGORITMO 1:**
    - pacotes desordenados:
        - não vai ocorrer, pois o origem só enviará um segundo pacote quando receber a confirmação do enviado antes
    - pacotes perdidos:
        - serão detectados pelo cliente ao estourar o tempo X sem receber confirmação.
    - pacotes repetidos:
        - pode ocorrer ou por ter havido uma repetição de roteamento ou porque o origem achou que não chegou (estourou o tempo ou porque a confirmação foi perdida). Ao receber um pacote duplicado (mesmo identificador anteriormente recebido) o destino deverá:
            - (a) CONFIRMAR: pois o origem pode estar esperando a confirmação
            - (b) descartar: porque já havia recebido.
        - Do lado do origem pode, igualmente, ocorrer duplicação da confirmação. Assim sendo, sempre que um origem receber a confirmação de um pacote N-1, estando esperando pela confirmação N, silenciosamente irá descartar e continuar esperando pela confirmação N.
    - mistura de pacotes de clientes distintos:
        - não vai acontecer pois o destino irá descartar pacotes que não vierem do origem com quem ele está conversando
    - concorrência:
        - não há. destino atende apenas um origem por vez.
        - Só está apto a atender outro depois que terminar a conversa com o atual (pelo recebimento do pacote “-”)
        - Um novo origem irá receber um 0,= e saberá que precisa esperar.
- PROBLEMAS:
    - transferência HALF-DUPLEX envia/confirma: péssimo desempenho
    - se origem abandonar a transferência sem enviar último pacote
    - se confirmação do último pacote for perdida, origem ficará reenviando ele eternamente.
    - se tempo de espera pela confirmação for curto demais ou longo demais
- **Algoritmo 2 (by Elgio):** neste procura-se resolver o desempenho do algoritmo 1
    - numeração de pacotes, como no algoritmo 1
    - origem não espera confirmação, destino é quem pede retransmissão
    - origem possui um *buffer* para os N últimos pacotes enviados, para o caso de retransmissões.
        - quando *buffer* encher, reutiliza a primeira posição em uma lista circular
        - origem sabe tratar pedidos de retransmissões do destino
    - Admite-se que quando o *buffer* encher já se passou tempo suficiente para o destino pedir a retransmissão do pacote do buffer 0
    - Formato de pacotes:
        - Envio de dados: "ID, +/-, DADOS"
        - Pedido de retransmissão: "ID, ?"
    - destino vai pedir a retransmissão do pacote K ao receber um pacote de ID maior que K, estando ele esperando o K.

    - Mistura de pacotes: descarta quem não é do origem esperado
- Lógica do origem:
    - define um buffer para N pacotes
    - Repete k iniciando em 1 ate k ser último
        - envia pacote k
        - armazena pacote k enviado na próxima posição livre do buffer
            - posição do buffer = ((k-1) mod N) + 1
            - sendo N o tamanho do buffer (buffer circular).
            - Exemplos para N = 10
                - k = 1, posição (0 mod 10) + 1 = 1
                - k = 10, posição (9 mod 10) + 1 = 10 (última)
                - k = 11, posição (10 mod 10) + 1 = 1 (primeira novamente)
        - Se destino pediu retransmissão
            - reenvia todos os pacotes da solicitação até o pacote k atual (inclusive)
                - Exemplo: k eh 7, destino pediu 5. Envia 5, 6 e 7
        - Se este era o último, sai do laco
        - Fim da transmissão do pacote k, volta para fazer k+1
- Lógica do destino:
    - aguarda primeiro pacote de algum origem
    - armazena Ip e porta origem
    - Repete (iniciando k=1)
        - Recebe um pacote X, supostamente de número k (ele espera o k)
            - Se não for do origem esperado, devolve erro (0,=) e descarta.
            - Se pacote X < k
                - Duplicidade de pacote. Descarta (ele ta esperando o 5 e chegou o 4)
            - Se X > K
                - FORA DE ORDEM (esperava o 1, mas veio o 2)
                - pede a retransmissão do pacote k que era esperado
                - descarta o pacote X que veio errado
            - Se X = K
                - K = K+1
                - Se K tinha marcação de último: ENCERRA Repete
            - Volta no repete para o próximo pacote
    - FIM DO LACO REPETE: Volta para aguardar transmissão de outro origem

- **AVALIAÇÃO ALGORITMO 2:**
    - pacotes desordenados:
        - serão considerados como perdidos, pois ao receber um pacote X > K vai descartar X e solicitar o reenvio de K
    - pacotes perdidos:
        - serão detectados pelo destino ao receber o pacote sem ter recebido o anterior.
    - pacotes repetidos:
        - pode ocorrer ou por ter havido uma repetição de roteamento ou porque houve um desordenamento (que o destino tratou com perda). Se o destino estiver esperando pelo K e receber o um X < K, sabe que é repetido e simplesmente descarta.
    - mistura de pacotes:
        - não vai acontecer pois o destino irá descartar pacotes que não vierem do origem com quem ele está conversando
    - concorrência:

        - não há. destino atende apenas um origem por vez.

- **PROBLEMAS do algoritmo 2**
    - melhora de desempenho SOMENTE se pacotes forem na ordem e sem perdas
    - pacotes desordenados serão considerados como PERDIDOS
        - pois o destino não criou um buffer para ele e não guarda pacotes recebidos fora de ordem.
    - Se houver muito desordenamento, acaba virando um HALF-Duplex novamente:
        - origem envia p1, p2 (perdido), envia p3
            - destino pede p2 ao receber o p3
        - envia p2, p3(perdido), p4
            - destino pede p3 ao receber o p4
        - Travado como na confirmação. Mas SOMENTE SE HOUVEREM MUITAS PERDAS OU DESORDENAMENTO.
    - PROBLEMA Sério:
        - origem envia último pacote e encerra, mas último pacote é perdido. E ai????
        - destino recebeu pN e espera Pn+1. Por quanto tempo? O que faz depois que destino julgar estar esperando demais? E se Pn+1 era o último, foi enviado e perdido de sorte que o cliente já encerrou sua execução?
        - Soluções?

Adicionando CONCORRÊNCIA ao UDP:

A) um mesmo servidor trata clientes distintos

- identifica cada cliente pelo par IP/Porta de Origem
    - pois DOIS clientes no mesmo IP deverão ter porta de origem distintas
    - E DUAS portas iguais de origem deverão vir de Ips distintos
- Para cada pacote, identifica qual o estado da "conversa"
- precisaria de uma tabela de clientes para isto e o estado de cada um
- Exemplo: se for uma ferramenta de *upload* de arquivos cada tabela teria
    - IP/Porta ORIGEM
    - nome do arquivo
    - último pacote recebido

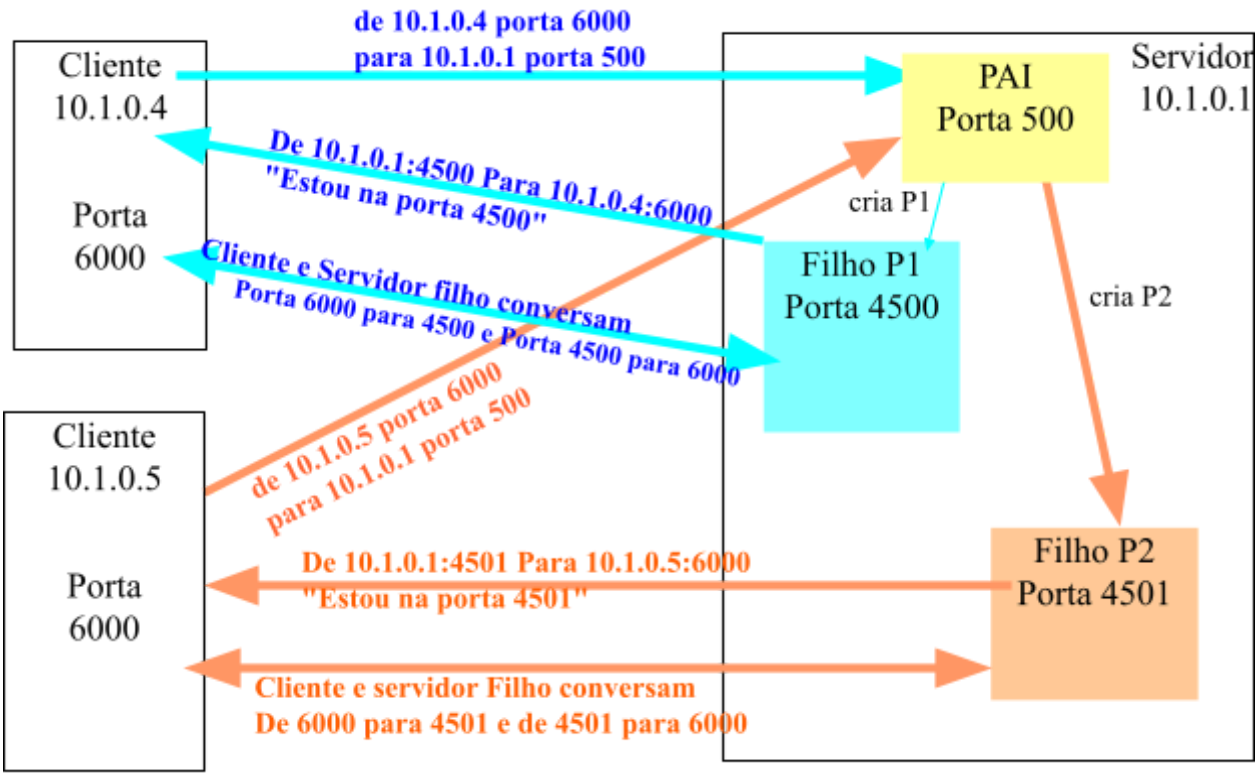
B) um processo novo (chama-se processo FILHO) para um cliente específico

- cada processo novo (filho) dedica-se a um único cliente
- Problema: apenas um processo pode ser dono da porta!
- Como os dados do cliente 1 seriam enviados para o FILHO 1?
- **Solução B1:**
    - Servidor cria processo filho
    - Servidor continua recebendo todos os pacotes para esta porta, mas mantém uma tabela de clientes e repassa os dados para o filho correspondente
    - Lógica do servidor:
        - Laço ESPERA pacote UDP
            - recebeu pacote
            - Obtém IP e Porta de origem
            - Ip/Porta já estão em sua tabela?
                - NÃO:
                    - cria um novo filho
                    - adiciona na tabela que aquele filho vai cuidar do cliente IP/PORTA
                    - envia pacote para o filho criado (comunicação local: pode até ser pela memória, por pipes, etc)
                - SIM: envia o pacote para o filho criado.
            - Filho notificou que encerrou: desaloca posição da tabela.
        - Volta para esperar novo pacote
        - Exemplo de dados desejáveis na Tabela de filhos:

Ip Origem	Porta Origem	PID DO FILHO
192.168.100.40	5000	2345
192.168.100.40	5001	2346
192.168.102.27	5000	2347

            - Se chegar um dado vindo de 192.168.100.4 porta 5000, servidor envia para o PID 2345.
            - Se chegar um dado vindo de 192.168.100.5 porta 5001, servidor cria um novo filho pois é um novo cliente.
    - Complexidade no gerenciamento:
        - manter tabela
        - manter filhos
        - desviar para filhos
        - forma de comunicação de servidor com filho

        - Em LINUX: Memória compartilhada, pipes, mensagens (sinais para acordar o filho, etc)
- **Solução B2:**
    - Cada filho tem uma porta nova para si
        - Problema: como o cliente saberá qual porta é?
            - Solução: filho conta para o cliente qual porta!
- Servidor cria processo filho para tratar
- processo NOVO pede uma nova porta ao SO
- processo FILHO informa ao cliente que a comunicação agora continua na porta X
- Demonstração na Figura:



- Tempo de reciclagem (em quanto tempo o servidor estará apto a receber outra requisição)
    - O servidor principal precisará apenas:
        - ao receber mensagem, criar filho
        - passar a mensagem para o filho
        - Voltar para esperar nova mensagem
    - Servidor principal recebe APENAS UM PACOTE
        - tempo apenas para criar o filho: aceitável
- lógica do servidor:
    - Repete (apenas DOIS passos):
        - Aguarda um primeiro pacote
        - cria um novo processo
- Lógica do Servidor FILHO
    - pede ao SO uma porta efêmera qualquer
    - obtém ip e porta de origem do cliente com quem devo conversar (herdado do pai)
    - Envio um pacote de confirmação ao cliente, informando que os dados devem vir na porta X (sendo X a porta que ganhou do SO)
    - Repete
        - recebe dados do cliente (se não for do meu cliente, descarta)
        - Recebeu o último pacote? MORRE

    - Poderia implementar apenas confiabilidade, mas sem concorrência pois este filho nasceu apenas para este cliente!
- Lógica do cliente
    - Envia um pacote de início (sem dados) para o servidor no IP dele e na porta do serviço
    - Aguarda um OK que virá de um servidor filho
    - Obtém a nova porta de destino que deverá ser usada (porta X)
    - Repete
        - envia dados para o servidor na porta X informada
        - respeita, se for o caso, algum protocolo de confiabilidade
- Se a aplicação for transmissora e receptora ao mesmo tempo?
    - Cada aplicação pode executar o protocolo de cliente e servidor
- Serviço de TFTP (*Trivial FTP*) executa sobre UDP e usa portas para filhos
- Mais fácil de implementar
    - sem tabelas de filhos
    - embora o número de conversas simultâneas esteja limitado ao número de portas UDP livres.
- Conclusões:
    - UDP não possui confiabilidade
        - e em nada ele me ajuda para que eu a implemente.
        - se desejar, é comigo, na minha lógica de programação
    - UDP não possui estrutura de dados para auxiliar a implementação de concorrência
        - e DENOVO em nada me ajuda para que eu a implementa
        - se desejar, é comigo, no braço e na minha criatividade enquanto programador
    - Implementar confiabilidade e concorrência exige mais esforço do programador
        - concorrência A: um mesmo servidor trata pacotes de acordo com o IP/Porta origem
        - concorrência B1: um novo processo trata os clientes, mas o servidor é quem despacha os pacotes UDP para o filho certo. Requer gerenciar tabelas de clientes e filhos que o atendem.
        - concorrência B2: Cada cliente gera um novo filho, filho e cliente se entendem em relação às portas que irão usar e livra o servidor de intermediar a comunicação.
            - Porque apenas um processo pode ser dono da porta de serviço
    - Confiabilidade e concorrência exige mais detalhes de implementação, como criação de *buffers*, se for o caso.
- se fosse necessário implementar uma aplicação confiável em UDP?
    - Qualquer detalhe deve ser tratado pela aplicação, usando os dados para sinalizar erros.
    - Problemas a serem resolvidos:
        - ordenamento de pacotes
        - replicação de pacotes
        - perda de pacotes
        - pacotes de clientes distintos
            - demonstrar no estudo de caso: enviando arquivos para o servidor.
        - concorrência (se for necessário)
            - O que é a concorrência?
            - Por que a necessidade da concorrência?

- **Pacote TCP**
    - exemplo de uma ligação telefônica
    - Garante recebimento
    - garante ordem no recebimento
        - Como garantir a ordem?
            - A) receptor confirma que recebeu, na falta desta confirmação, transmissor reenvia por *time out*
            - B) ou receptor **solicita** que seja retransmitido algum que faltou
    - O caso do TCP é semelhante ao A:
        - O remetente espera confirmação de recebimento

        - se não receber, torna a reenviar
      - Estabelecimento *de time outs*
    - *Handshake:* demonstrar

- Definição do servidor:
  - Vai esperar uma conexão
  - Porta no modo PASSIVO
- Definição do Cliente
  - Vai INICIAR uma conexão (ou seja, fará o *handshake*)
  - Porta no modo ATIVO (de ação, tomar a iniciativa)
- Cabeçalho TCP:

Cabeçalho TCP

- **Dados do Cabeçalho TCP**
  - Porta Origem: 16 bits
  - Porta Destino: 16 bits
  - Número de sequência: 32 bits (tem a ver com as confirmações)
  - Número ACK: 32 bits (tem a ver com as confirmações)
  - HLen: 4 bits (Tamanho do cabeçalho em Words).
  - Reservado para uso futuro: 6 bits (os dois últimos bits são flags adicionais)
  - Flags: 6 bits (algumas ferramentas mostram 8 bits, juntando os dois do reservado)
    - Os flags determinam o tipo de pacote
    - URG: pacote possui dados urgentes
    - ACK: Acknologe
    - PSH (Push): Descarregamento do buffer
    - RST: Resetar/Abortar sessão
    - SYN: *Synchronize sequence Number.*
    - FIN: Finalização da conexão
  - Janela: 16 bits (sua função será descrita posteriormente)
  - *Checksum*: 16 bits
  - Ponteiro para dados urgentes: 16 bits
- Explicação do *handshake* (SYN -> SYN+ACK -> ACK)
  - uso dos flags

- Confiabilidade implementada pelo TCP
  - baseada em confirmações
  - Confirmação enviada na forma de próximo byte esperado
  - uso de Syn e Acks
  - Cada máquina escolhe um número sequencial inicial
    - Este número será a primeira posição do seu *buffer* de envio
    - No primeiro SYN, envia o seu número no campo sequência
  - Recebe o número de B no campo sequência e o seqA + 1 no de ACK.

        - Este número vai ser usado para a primeira posição do *buffer* de recebimento
    - Confirmações podem ser cumulativas
- Uso de janelas deslizantes, com SYN e ACK
    - utiliza uma janela de tamanho conhecido
    - transmite todos da janela sem precisar receber confirmação
    - janela anda a medida que recebe confirmações
- Questões quanto ao tamanho da janela
    - Janela pequena:
        - travado, pouca memória
    - Janela grande:
        - Mais rápido (menos travado)
        - muita memória
        - problema do efeito cascata
            - perda de um pacote causa retransmissões desnecessárias
    - Janela = 0 => PAUSE

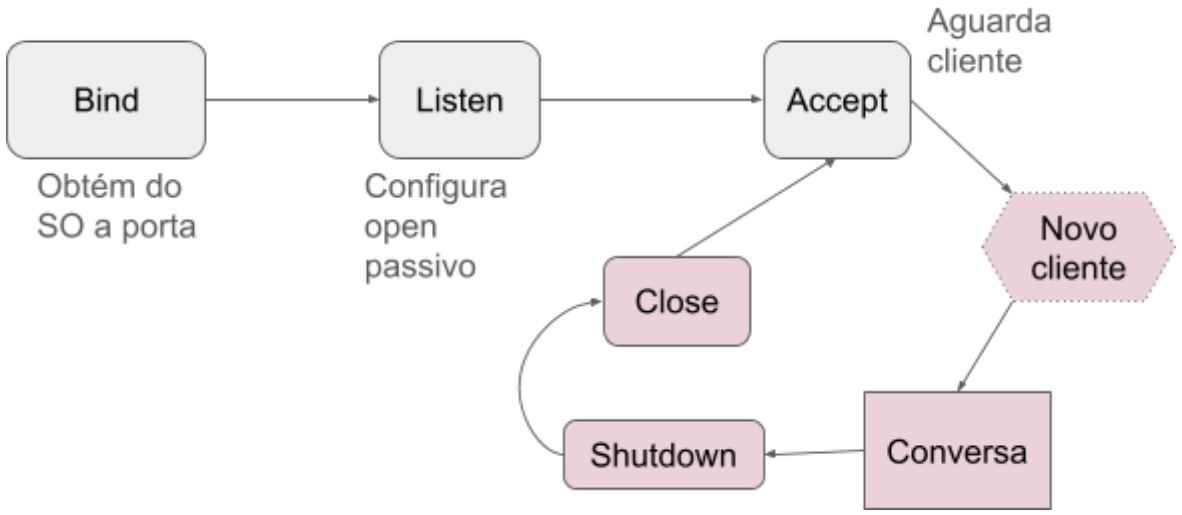
- Exemplo de uma conexão TCP cliente servidor
    - com o uso das janelas
    - com números sequenciais aleatórios no início
    - exemplo de uso do PUSH
    - Retransmissão cumulativa

- **Programação em TCP**
  - Código tcpserv.c
    - bind, listen
    - accept
      - fica preso até que chegue uma conexão
      - recebe socket de conexão
  - É a pilha quem faz ordenamentos e o *handshake*

Concorrência no TCP

- A pilha dá um socket de conexão para cada novo cliente na porta
- amarra ip e porta de origem do cliente a este socket de conexão
- um filho pode ler e escrever neste socket de conexão
- tcpservConc.c (comentário sobre o fork)

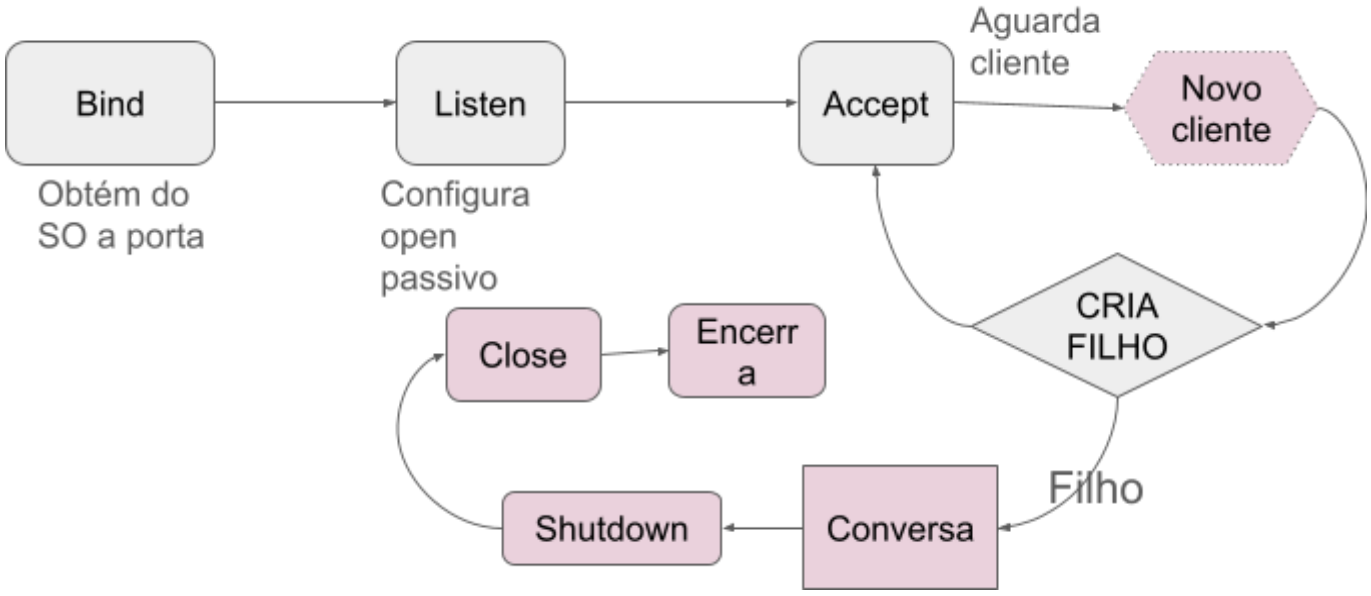
◦ Socket ouvinte e Socket de conexão
◦ desenhar estrutura do SO para ouvinte e para conexão
  - tabela: um socket para CADA conexão



◦ Problema: servidor fica preso no accept
  - Solução: passar socket de conexão para um filho

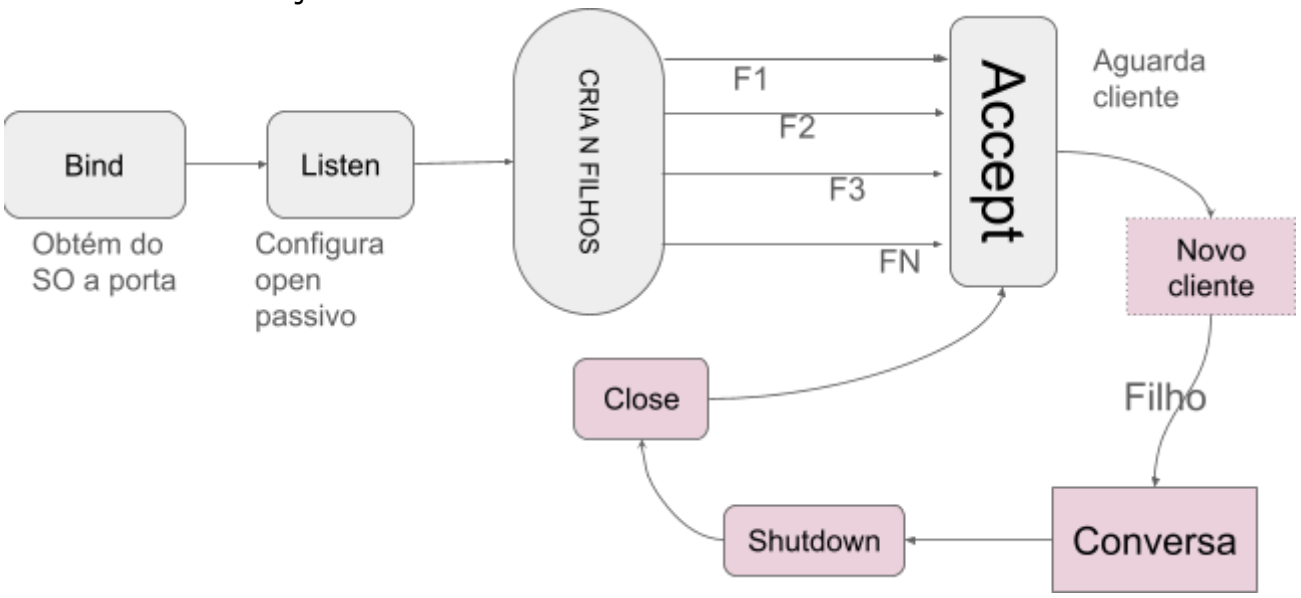
1) Concorrência PÓS accept (concorrência simples)

- número indeterminado de conexões
- Lento pois precisa criar os filhos antes de conversar
- Tempo de inatividade alto
- Muito fácil de programar (nada complexo)
- Viável para servidores pequenos



2) Estouro da manada

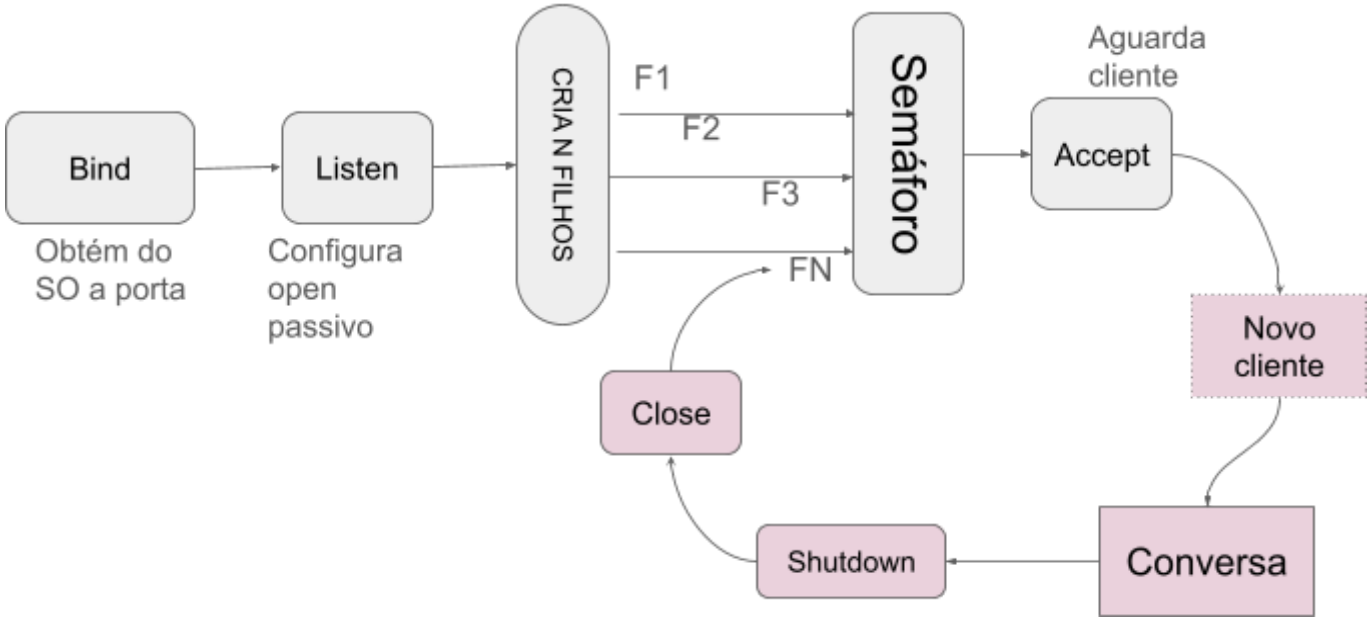
- Pré criação de N filhos
  - Todos presos no *accept*
- Fácil de programar
  - não requer maiores complexidades de gerenciamento
- Problema: número de filhos FIXO
  - só pode atender no máximo ao número de filhos já existentes
- Tempo de inatividade pequeno (BOM)
- Nem todos os Sistemas Operacionais suportam
  - pois depende de uma propriedade da chamada de sistema *accept*
  - BSD e Linux funcionam
  - Demonstração



3) Bloqueio no *accept*

- Semelhante ao anterior

- mas evita o "estouro"
- compatível com todos os sistemas operacionais
- bloqueia-se os filhos antes que eles executem *accept*
- usa-se algum sistema de bloqueio que o Sistema Operacional possui
  - como semáforos ou mutex em Linux
- Programação razoavelmente fácil (depende do sistema de semáforos existentes)
- Mesmo problema de conexões limitadas



4) Gerenciamento de filhos

- Criar os filhos antes, mas o pai executa o accept
  - Ele recebe cliente e repassa o socket para um filho disponível
- Filho aguarda cliente do pai
  - requer comunicação entre processos, coisa que todo SO possui
- Filho, ao terminar, NÃO MORRE, mas avisa o PAI que voltou a estar disponível
- Pai precisa manter relação de filhos ocupados e livres
- Pai pode decidir:
  - criar mais filhos, pois estão quase todos ocupados
  - matar alguns filhos, pois tem poucas conexões e muitos filhos
  - Rápido, porém complexo
- Usando *Threads*
  - Qualquer método anterior pode suportar *Threads*
    - Mais rápido criar *threads*
    - menos memória ocupada
    - Porém mais difícil de programar

Solução de MELHOR DESEMPENHO:

- Algoritmo 4 (gerenciamento) usando *Threads*
- Cria-se *Threads* anteriormente
- cada uma bloqueada esperando que o PAI a acorde
- Pai:
  - executa accept
  - escolhe Thread filha para atender
  - volta a executar accept
- *Threads* Filhas:

    - Acordada pelo Pai
    - conversa com cliente
    - informa Pai que voltou a estar disponível
- Periodicamente um nonitor:
    - verifica índice de ocupação de *Threads*
        - MUITAS? Mata algumas
        - POUCAS? Cria mais

Nat dinâmico: Mascaramento

- NAT 1:N: Apenas um único IP para toda a rede
- Como?
- Associa uma conexão (socket) a uma porta local do servidor
    - também chamado de PAT (*Port Address Translation*)
- Tabela mais complexa:
- Passos:
    - Cliente escolhe porta alta (2000)
    - Inicia conexão
    - NAT abre outra porta auto LOCAL
        - procura usar a mesma do cliente, se estiver disponível
    - Associa esta porta a conexão (TABELA)
    - Traduz trocando Porta e IP
    - Mostrar exemplos.
- Técnica chamada de Mascaramento de IP
- Mascaramento IMPOSSIBILITA uso como servidor
    - Pois não existe tabela para a máquina.
- CGNAT
    - NAT 1:N (Dinâmico) feito pelo provedor
    - Um único IP público compartilhado por vários clientes
        - e, provavelmente, cada cliente já compartilha com sua rede. duplo NAT
    - Problemas do CGNAT
        - qual ip atribui aos clientes?
            - um da faixa dos privados? 10.0.0.0/8 por exemplo?
            - Não, pois daria conflito se um dos clientes usasse a mesma faixa
            - Reserva da faixa 100.64.0.0/10 (RFC 6598)
        - Segurança

IPV6

- tipo 86DD
- Aumenta de 32 bits para 128 bits de endereçamento
- Muda a forma de representação, passando a ser hexa decimal
  ```
  32.0.69.86.16.16.31.48.18.52.69.96.0.255.222.2
  2000:4556:1010:1F30:1234:4560:00FF:DE02
  ```
- Alternativas para minimizar a representação: supressão de 0's:
  ```
  0000:0000:0000:0000:0034:4560:00FF:DE02
  ::34:4560:FF:DE02
  ```
- Cabeçalho de tamanho único (10 *words* SEMPRE)
  - Menos campos que o do IPv4
  - com possibilidade de cabeçalhos de extensão.
  - Um cabeçalho de extensão precisa ter o código do tipo do próximo cabeçalho
  - inclusive para o nível de transporte
    - É tratado como cabeçalho de extensão
  - Cabeçalhos de extensão não precisam ser tratados pelo roteador.
- Fragmentação só na origem
- Comunicação
  - *unicast*
  - *anycast* ou cluster:
    - para alguém, o mais próximo (ou mais disponível)
    - permite que vários equipamentos compartilhem o mesmo IP
  - *multicast*
  - inexistência de *broadcast.*
    - *broadcast* deve ser emulado por *multicast*
- Vários ips por interface.
  - Atualmente isto podia ser realizado com interfaces virtuais
  - Porém, o Ipv6 explora bem uma característica do protocolo SCTP (sucessor do TCP) o de balancear uma mesma conexão entre várias interfaces
- Arp incorporado ao ICMPv6
- Espaço de endereçamento:
  - Ipv4: Duas hierarquias
    - bits para rede
    - bits para host
  - IPV6: Várias hierarquias
    - 8 bits iniciais: TIPO
      - 0000 0000 = Compatibilidade com IPv4
      - 1111 1110 = Uso local (PRIVADO)
      - 1111 1111 = *Multicast*
      - 010: Tipo endereçamento de Provedor
      - 100: Tipo endereçamento geográfico
    - Demais bits: outras hierarquias.
    - Exemplo:
      ```
      010 <ID Provedor> <ID assinante> <ID SubRede> <HOST>
      ```
  - Sugestão de 64 bits para host
    - pode-se usar o MAC da placa de rede para endereço de host
    - formato EUI-64

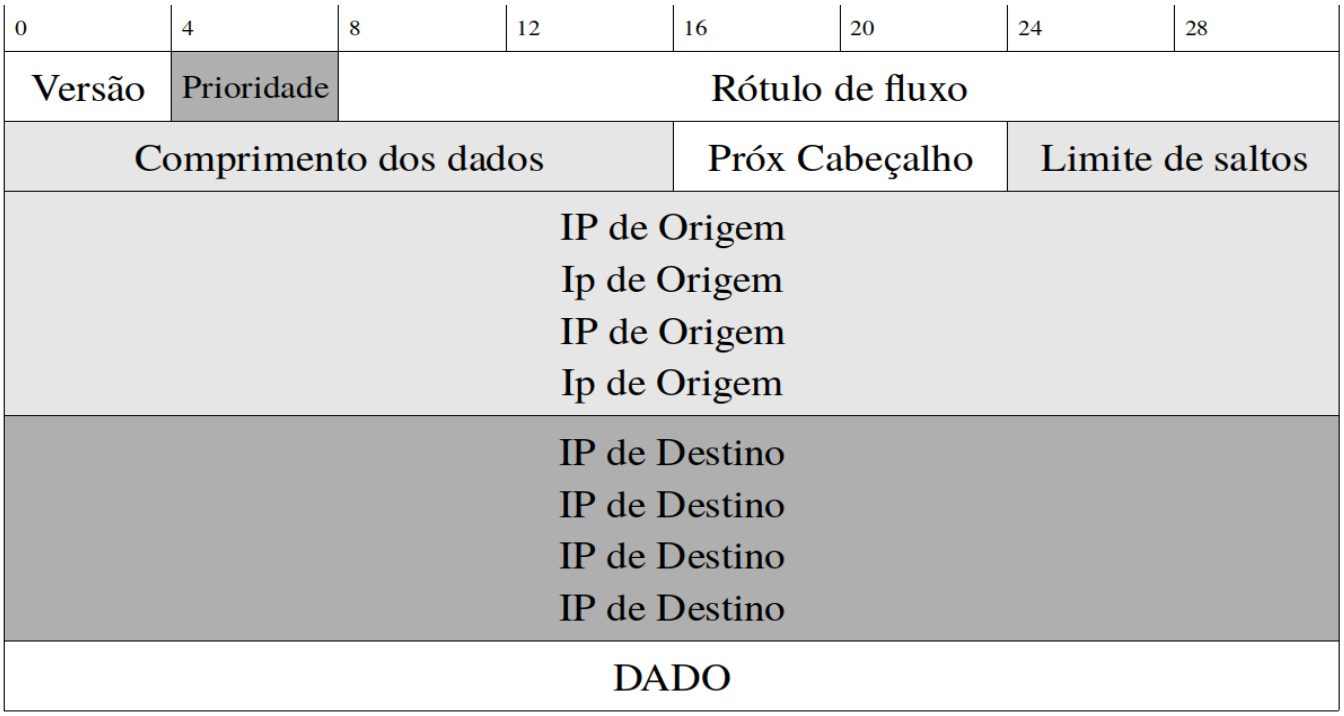
- Compatibilidade de endereçamento com IPV4:
  - 80 bits iniciais em ZERO
  - 16 bits subseqüentes em ZERO (obsoleto) ou em UM
  - 32 bits finais idem IPv4.
  - Exemplo:

```
IPv4: 143.54.11.5
IPv6: 0:0:0:0:0:FFFF:143.54.11.5
ou    ::FFFF:143.54.11.5
ou    0:0:0:0:0:0:143.54.11.5 (obsoleto)
ou    ::143.54.11.5 (obsoleto)
```

  - Formato IPv4 atual: `::FFFF:w.x.y.z`

- Roteamento no Ipv6
  - Facilitada pela divisão do Ip
  - Roteadores mais leves, com menos tarefas
  - Fragmentação apenas fim-a-fim
    - Roteador JAMAIS fragmenta um datagrama Ipv6
    - Cabe ao origem descobrir o MENOR MTU existente na rota e usar este
      - Na prática isto já é feito por algumas implementações do Ipv4
      - feito com pacotes ICMP
      - Se não descobrirem MTU mínimo, usar 576 (MENOR MTU admitido)
      - Controle de fragmentação é feito por cabeçalho de extensão.
    - Roteadores mais leves:
      - sem cálculo de CheckSUM (mesmo ao decrementar saltos)
      - sem fragmentação
      - Sem cabeçalho com tamanho variável
      - Apesar de 128 bits contra 32, o trabalho do roteador está BEM MAIS LEVE

- Cabeçalho do IPV6:



  - Versão (4 bits): 0110 para Ipv6
  - Prioridade (4 bits)

        - Semelhante ao Tipo de Serviço do Ipv4
        - Para determinar prioridade dos pacotes quando estiver congestionado
    - Rótulo de fluxo (24 bits):
        - Controles de fluxos especiais
        - usado para áudio e vídeo de tempo real
    - Comprimento dos dados:
        - Tamanho dos dados
        - Sem necessidade de tamanho do cabeçalhos, como no Ipv4, pois cabeçalho Ipv6 tem sempre o mesmo tamanho
    - Próximo cabeçalho:
        - identifica o tipo do próximo cabeçalho
        - Usado para cabeçalho do transporte (TCP, UDP, etc)
        - Para cabeçalhos estendidos do próprio Ipv6 (semelhante ao opções do Ipv4)
    - Limite de saltos:
        - Como o TTL, exatamente igual
        - Só mudou o nome para saltos
    - Ip origem: 128 bits
    - Ip origem: 128 bits
    - Não tem soma de verificação (*CheckSum*)
- Cabeçalho Ipv4 X Ipv6
    - Ipv4:
        - 12 campos, 5 a 15 words de cabeçalho
        - possibilidade de opções
    - Ipv6:
        - SEMPRE 8 campos (ou 7) e tamanho fixo de 10 Words
        - Cabeçalhos de extensão, se necessário

Aula 15 - 12/Nov/2025

IPv6

- endereçamento
- atribuição
- regras de migração
    - NAT64 + DNS64

Aula 16 - 19/Nov/2025

APLICAÇÃO:

- Aplicações:
  - DNS, SMTP [RFC 2821], HTTP [RFC 2616], etc
- Portas como designador de serviços:
  - 22 - SSH 25 - SMTP 80 - HTTP 53 - DNS 443 - HTTPS

DNS - Domain Name Service

- Antes era em arquivos
- Serviços de nomes (DNS): porta 53 UDP e TCP
  - Em hierarquia
    - Servidores RAIZ: são 13 nomes de DNS replicados pelo mundo
    - Sua consulta é roteada para o servidor mais próximo
      - Se IPv4, é por gambiarra
      - Se IPv6, é usando o modo anycast
  - Realiza *cache:* Questões quanto ao tempo de *cache*
  - Tipos de consultas:
    - recursivo: pergunto para meu servidor de cache e ele procura
    - interativo: "navega" na árvore desde o root até o NS do domínio

- Tipo de servidores DNS
  - cache
  - autoritativo master
    - responsável por um domínio
  - autoritativo slave
    - corresponsável por um domínio
    - **baixa zonas completas do *master***
  - Nomes especiais:
    - MX: Para perguntar quem é o servidor de email (Mail eXchange) do domínio
    - NS: *Name Server:* para perguntar quem é o servidor de DNS do domínio
    - TXT: texto livre. Literalmente qualquer coisa

Brincando com DNS. Comando nslookup Linux/DOS:

Quem é o servidor de emails que atende por @inf.ufrgs.br?

```
$ nslookup -type=MX inf.ufrgs.br
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
inf.ufrgs.br    mail exchanger = 0 smtp-gate.inf.ufrgs.br.
```

Resposta: é a máquina que se chama smtp-gate.inf.ufrgs.br, cujo número IP é 143.54.11.20. Se eu desejar enviar um email para algum usuário do domínio inf.ufrgs.br, é com a porta 25 desta máquina que eu devo falar.

Exemplo de configuração de DNS (tabela livre de sintaxe)

- Domínio redes.xyz (fictício)

1. tenho DOIS servidores com IP público
   - Exemplo: 143.54.11.5 e 143.54.11.6
2. Configuro um como sendo master (143.54.11.5)

   - Entradas em forma de tabela

Domínio redes.xyz			
Nome	**Tipo**	**Valor**	**TTL**
	NS	ns1.redes.xyz.	3600
	NS	ns2.redes.xyz.	3600
	MX	email.redes.xyz.	3600
ns1	A	143.54.11.5	3600
ns2	A	143.54.11.6	3600
email	A	143.54.11.10	3600
WWW	A	143.54.11.15	3600
WWW2	CNAME	www	3600

Tabelas presente tanto no 143.54.11.5 quanto no 143.54.11.6. **Só no primário é editado**

3. 143.54.11.6 será slave, terá a mesma tabela, mas a irá copiar do 143.54.11.5
4. Compro o domínio redes.xyz e registro 143.54.11.5 e 143.54.11.6 como NS deste domínio

   - DNS Reverso
     - Obtém um nome a partir de um IP
     - Muito útil por questões de segurança
     - Exemplo:
       - Qual o Numero Ip de www.inf.ufrgs.br?
         - Em linha de comando Linux/Windows:

```
$ nslookup www.inf.ufrgs.br
   Name:    www.inf.ufrgs.br
   Address: 143.54.11.34
```

       - Qual o nome do IP 143.54.11.34
         - Em linha de comando:

```
$ nslookup 143.54.11.34
   www.inf.ufrgs.br.
```

   - Como direto e reverso CONFEREM, pode-se admitir que é uma máquina idônea

Serviço HTTP

- Possui cabeçalho e comandos. Exemplo:

```
http://www.inf.ufrgs.br/~elgios/linux/programacao_C/
      |< ENDERECO DNS >|<Arquivo dentro do servidor>|
```

- Principais comandos:
  - GET : obter dados. Exemplo: (conectado em www.inf.ufrgs.br)
    GET /~elgios/linux/programacao_C/ HTTP/1.0

- Problema: seria desejável que uma única máquina servisse vários domínios
  - Exemplo: elgio.prof.nom.br e www.redes2.info

`http://elgio.prof.nom.br/lixo.html`

Equivale:

```
telnet elgio.prof.nom.br 80
GET /lixo.html HTTP/1.0
```

`http://www.redes2.info/lixo.html`

Equivale:

```
telnet www.redes2.info 80
GET /lixo.html HTTP/1.0
```

  - E se o objetivo era fazer sites diferentes e no www.redes2.info não existisse o lixo.html?
  - Como o servidor vai saber a qual dos domínios se está conectando, já que é o mesmo servidor?
  - HTTP versão 1.1 impôs o campo HOST no cabeçalho:

`http://elgio.prof.nom.br/lixo.html`

Equivale:

```
telnet elgio.prof.nom.br 80
GET /lixo.html HTTP/1.1
HOST: elgio.prof.nom.br
```

`http://www.redes2.info/lixo.html`

Equivale:

```
telnet www.redes2.info 80
GET /lixo.html HTTP/1.1
HOST: www.redes2.info
```

    - Desta forma, uma máquina pode servir a VÁRIOS domínios e saber em qual diretório estão os arquivos de cada domínio

Aula 17 - 26/Nov/2025

- Ips dinâmicos: RARP, BOOTP
- DHCP Dynamic Host Configuration Protocol
  - Não apenas IP, mas várias outras informações:
    - Ip/Máscara
    - Tempo de expiração (típico de 2 horas)
    - Ip do GW
    - Ip(s) do(s) DNS
    - nome do host
    - muitas outras...
  - Usa UDP para solicitar um IP
    - Mas COMO? Como gerar um pacote UDP se não tem IP?
    - Envia para 255.255.255.255
      - sabidamente TODOS de sua rede, mesmo sem saber que rede
    - E a resposta?
      - ou o servidor envia para 255.255.255.255
        - Todos recebem
        - Pelos dados, somente quem requisitou considera a resposta
      - ou envia direto para o MAC do solicitante
        - Para isso precisa inserir manualmente o IP atribuído a tabela ARP
        - Pois senão, a camada de Rede vai fazer um ARP e ninguém vai responder.
  - Toda a confiabilidade é do cliente
    - time outs: se não receber um ip:
      - Espera aleatoriamente de 0 a 4 segs
      - Dobra o tempo a cada nova tentativa, até 60 seg, quando volta a ser aleatório
      - Isto para evitar sobrecarga do servidor em caso de muitas máquinas ligarem ao mesmo tempo e pedirem ip em rajadas
    - cliente deve solicitar uma renovação, geralmente quando o tempo estiver a 50%
    - se cliente não solicitar, azar

- Configuração do DHCP
  - O administrador pode dar uma faixa de ips que o DHCP irá gerenciar dinamicamente
  - Adicionalmente pode FIXAR Ips por MACs
    - Estes ips não devem estar conflitando com os dinâmicos
- Ideal que exista UM servidor por rede
  - Se existirem DOIS, o cliente poderá receber DUAS atribuições para uma mesma requisição (ou mesmo N)
  - Sem problemas para o cliente, pois o DHCP permite que ele escolha a que melhor lhe convém (a primeira?)
  - Poderá haver PROBLEMA se os servidores atuarem em uma mesma faixa de ips.
- Um DHCP por rede
  - Ou DHCP relay
  - Roteadores comerciais possuem Relay (ou até mesmo o próprio DHCP)