Redes de Computadores

Protocolos de Transporte na Internet

User Datagram Protocol - UDP

NAT/NAPT

Introdução a protocolos de aplicação (sockets)

Aula 24

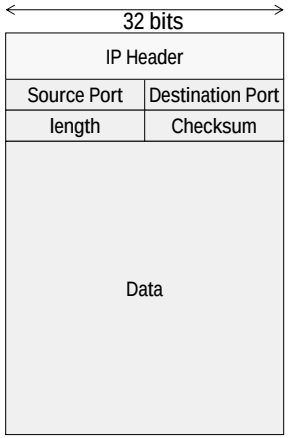
Introdução

- Entidades da camada de transporte oferecem serviços não orientados a conexão
  - Apesar de não ter garantias de entrega, de ordenamento e de não duplicação eles tem o seu valor.
  - Na arquitetura TCP/IP corresponde ao protocolo UDP
- Os serviços das entidades de transporte são disponibilizados para a camada de nível superior
  - Na arquitetura TCP/IP isso é feito através da interface de *sockets*
- Funcionamento do NAPT (vulgarmente conhecido como NAT)
  - Envolve itens da camada de rede e da camada de transporte

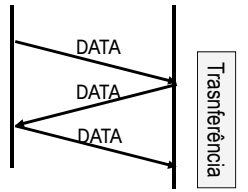
User Datagram Protocol (UDP)

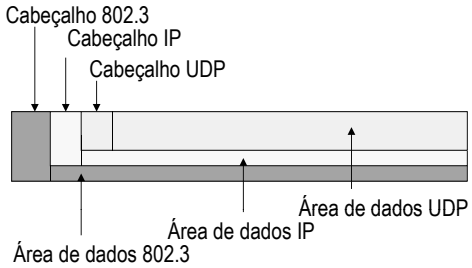
- Descrito na RFC 768
- PDU do UDP é denominada datagrama
  - Orientado a mensagem
  - Encapsula uma mensagem da aplicação sem realizar fragmentação
    - Controle é do próprio processo usuário
- Protocolo de transporte da família TCP/IP não orientado à conexão
  - Por não executar controle de fluxo, de erro e de ordenamento, possui baixo custo de processamento
  - Em relação ao IP agrega funcionalidade de multiplexação e demultiplexação
    - Comunicação processo a processo
- Processos origem e destino são identificados através de portas

Formato do datagrama UDP (relembrando...)



PDU – *Protocol Data Unit*





Campos do datagrama UDP

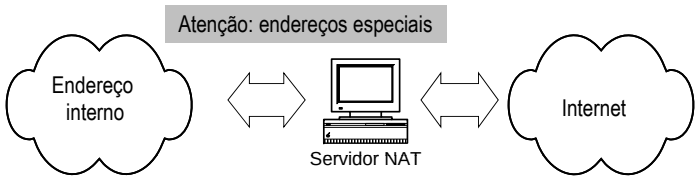
- Porta fonte
  - Associada ao processo de origem (multiplexação)
  - Permite ao destino retornar mensagens ao processo de origem
- Porta destino
  - Usada para demultiplexação das mensagens encapsuladas nos datagramas
- Tamanho
  - Tamanho total do datagrama UDP (inclui cabeçalho + dados)
- *Checksum*
  - Verificação da integridade dos dados (complemento de 1 em 16 bits)
  - Calculado sobre um pseudo-cabeçalho (IP destino, IP fonte, campo protocolo, tamanho TPDU e a constante zero), o cabeçalho UDP e os dados
- Dados

Razões para usar UDP

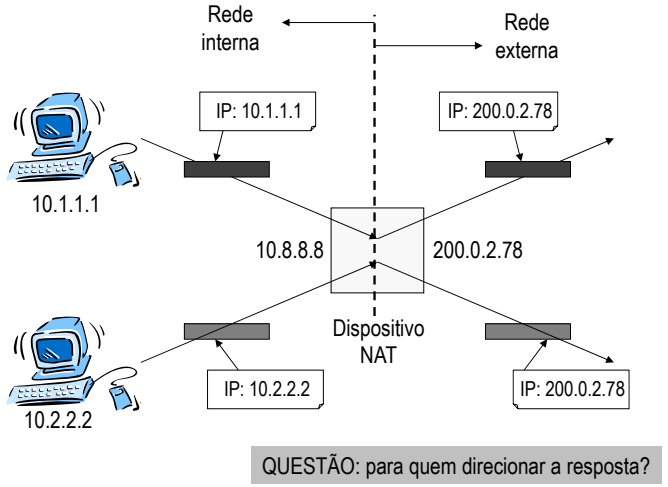
- Menor custo computacional
  - Não há estabelecimento de conexão, nem estados a monitorar e gerenciar
  - Não faz controle de perda, repetição, ordenamento e congestionamento
    - Usado em aplicativos que toleram erros de entrga
  - Cabeçalho simplificado
- Melhor controle dos dados enviados
  - Dados da aplicação são empacotados pelo UDP e enviados ao IP
    - Cada mensagem aplicação deve caber em um datagrama UDP
      - Desejável que caiba em um datagrama IP para evitar fragmentação
- Uso de *broadcast* ou *multicast*

NAT- *Network Address Translation*

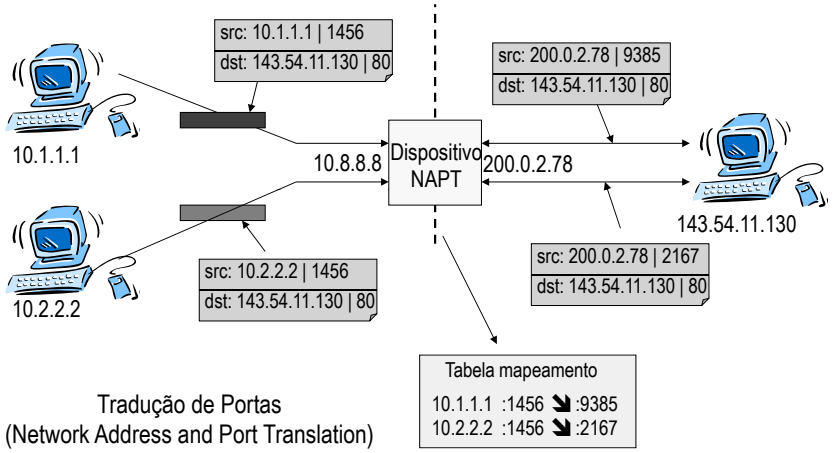
- Endereços IPs devem ser únicos
- Constatação: nem toda máquina precisa ter endereço Internet válido
  - Para que desperdiçar endereços IP válidos?
- Endereços IP reservados para redes não conectadas (RFC 1918)
  - Bloco/Classe A: 10.0.0.0 – 10.255.255.255/8 (16.777.216 IPs)
  - Bloco/Faixa de classe B: 172.16.0.0 - 172.31.255.255/12 (1.048.576 IPs)
  - Bloco/Faixa de classe C: 192.168.0.0 - 192.168.255.255/16 (65.536 IPs)



Funcionamento do NAT



Funcionamento do NAPT (*Cont.*)



Tradução de Portas
(Network Address and Port Translation)

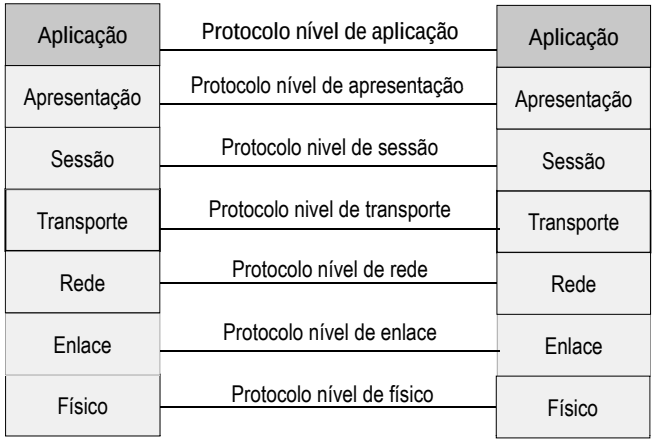
Traduções de endereços, portas e protocolos

- NAT (RFC 1631)
  - Mapeamento de um endereço IP em outro
  - Existem variedades: N:1, M:N; 1:1
- NAPT (RFC 3022)
  - Mapeamento de endereços e portas em outros endereços e portas
  - Denominado corriqueiramente de “NAT” (ou NAT tradicional)
- NAT-PT (RFC 2766)
  - Mapeamento de endereços e portas em outros endereços e portas
  - Capacidade de traduzir protocolos da camada de rede (*Protocol Translation*)
    - Um dos mecanismos previstos para a transição IPv4-IPv6

Críticas ao NAT/NAPT

- Viola regra do endereço IP ser único
- Viola a independência de camadas
  - Mistura informações nível 3 (end. IP) com informações do nível 4 (porta)
- Só funciona com protocolos TCP e UDP (porta)
- Transforma a Internet (rede orientada a pacotes) em uma rede orientada a conexão (circuito virtual)
  - Necessidade de manter estado A-B; B-C para comunicação A-C
- Aplicações que inserem o IP ou porta no corpo de suas mensagens podem apresentar disfunção com o NAT
  - Ex.: FTP, H.323 e IPsec

Nível de aplicação



Principais protocolos de aplicação da Internet

- DNS
- HTTP
- HTTPS
- FTP
- TFTP
- SMTP
- POP3
- IMAP
- TELNET
- SNMP
- BOOTP
- DHCP
- NNTP
- NFS
- SSL

Visão da camada de transporte pela camada aplicação

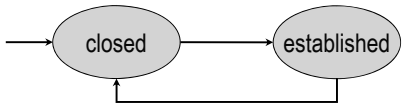
- Interface de *sockets*
  - Interface entre a aplicação e as entidades de transporte
  - Conjunto de primitivas para definir portas, estabelecer conexão (ou não), enviar e receber dados
- Originalmente proposta para ambientes Unix e linguagem C
  - Adotada em várias plataformas e linguagens (java, C#,...)
  - Concebido para ser genérico
    - A Internet corresponde a família AF_INET
- Construída sobre a abstração de *socket*
  - Na Internet um *endpoint* é representado pelo par (endereço IP e porta)
  - No TCP, cada conexão é identificada por um par de *endpoints* (*socket pair*)

*Endpoint*s

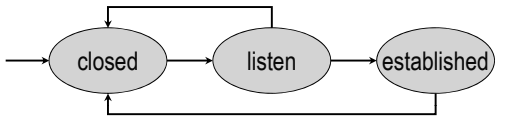
- Cliente e servidor possuem *endpoint* local e remoto
- *Endpoint* local
  - Criado, por *default*, com o endereço IP especial 0.0.0.0 e uma porta arbitrária
  - Pode ser atribuído uma porta e endereço IP específicos, tipicamente:
    - Servidor: configura uma porta específica
    - Cliente: porta (efêmera) selecionada pelo sistema operacional
  - Pode participar de várias comunicações com *endpoints* remotos distintos
- *Endpoint* remoto
  - Criado, por *default*, com endereço IP especial 0.0.0.0 e porta ANY (*)
  - Pode ser atribuído um endereço IP e uma porta específica
    - Cliente : deve especificar endereço IP e a porta do servidor
    - Servidor: preenche com endereço IP e porta provenientes do cliente

Estados de um *socket* (serviços orientados a conexão)

- *Socket* ativo: cliente
  - Usado para solicitar requisições de conexão ao servidor



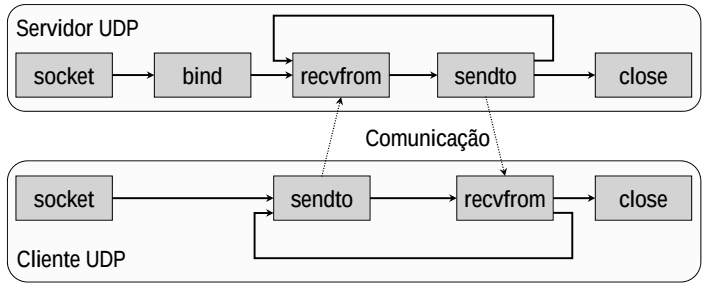
- Socket passivo: servidor
  - Empregado pelo servidor para aguardar pedidos de conexão



Interface de *sockets*

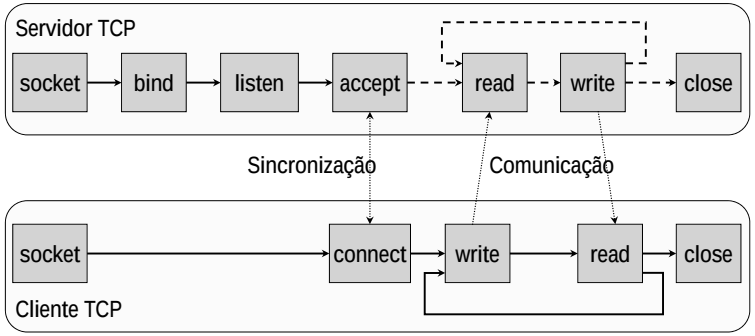
- Socket é um descritor de arquivo
  - Paradigma abrir-ler-escrever-fechar
- As primitivas básicas são:
  - socket( )
  - bind( )
  - listen( )
  - accept( )
  - connect( )
  - write( ), sendto( )
  - read( ), recvfrom( )
  - close( )

Clientes e servidores UDP



Não há estados em sockets UDP (serviço sem conexão)

Clientes e servidores TCP



Socket original (*listen*)
Socket conectado (criado no *accept*)

Exemplo *sockets* (lado servidor TCP)

```
main() {
  int cont,create_socket,new_socket,addrlen;
  int bufsize = 1024;
  char *buffer = malloc(buffsize);
  struct sockaddr_in address;

  printf("\x1B[2J");
  if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
    printf("The socket was created\n");
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(15000);
  if (bind(create_socket,(struct sockaddr *)&address,sizeof(address)) == 0)
    printf("Binding Socket\n");
  listen(create_socket,3);
  addrlen = sizeof(struct sockaddr_in);
  new_socket = accept(create_socket,(struct sockaddr *)&address,&addrlen);
  if (new_socket > 0){
     printf("The Client %s is connected...\n",inet_ntoa(address.sin_addr));
     for(cont=1;cont<5000;cont++)
  printf("\x7");
 }
```

```
do{
   recv(new_socket,buffer,bufsize,0);
   // do service...
   send(new_socket,buffer,bufsize,0);
   ....
}while(strcmp(buffer,"/q"))
close(new_socket);
close(create_socket);
}
```

Exemplo *sockets* (lado cliente)

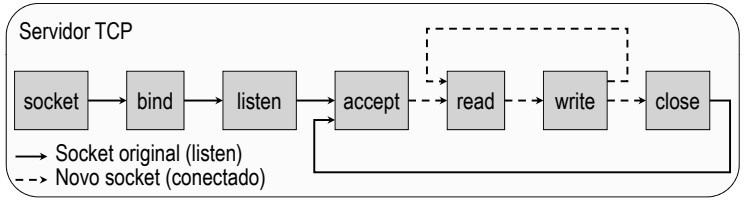
```
main(int argc,char *argv[]) {
  int create_socket, buffsize = 1024;
  char *buffer = malloc(buffsize);
  struct sockaddr_in address;

  if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
       printf("The Socket was created\n");
  address.sin_family = AF_INET;
  address.sin_port = htons(15000);
  inet_pton(AF_INET,argv[1],&address.sin_addr);

  if (connect(create_socket,(struct sockaddr *)&address,sizeof(address)) == 0)
        printf("The connection was accepted with the server %s...\n",inet_ntoa(address.sin_addr));
  do{
     gets(buffer);
    send(cria_socket,buffer,bufsize,0);
    recv(create_socket,buffer,bufsize,0);
    printf("Message recieved: %s\n",buffer);
    if (strcmp(buffer,"/q"))
         printf("Message to send: ");
  }while (strcmp(buffer,"/q"));
  close(create_socket);
}
```

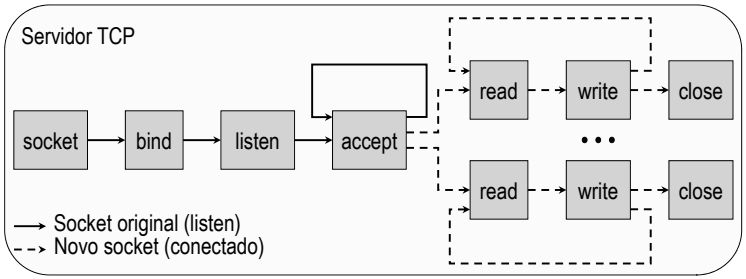

Servidor iterativo (*single threaded*)

- Adequado para serviços com reduzida taxa de requisições e serviços cujas requisições possuem baixa carga de processamento



Servidor concorrente (*multithreaded*)

- Serviços com elevada taxa de requisições e serviços cujas requisições possuem alta carga de processamento
- Implementado com várias threads ou processos independentes



Funcionamento de *sockets*: lado servidor

- *Socket* (único) que atende o serviço
  - Sempre em estado ***listen*** (processam segmentos SYN)
  - *Endpoint* local possui o endereço especial 0.0.0.0 e porta específica do servidor
  - *Endpoint* remoto possui o endereço especial 0.0.0.0 e a porta ANY (*)
- *Sockets* criados para atender requisições dos clientes
  - Estado ***established*** (processam segmentos de dados)
  - *Endpoints* locais possuem o endereço IP e porta específica do servidor
  - *Endpoints* remotos possuem os endereços IP e as portas dos respectivos clientes

> netstat -tan

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:25	0.0.0.0:*	LISTEN
tcp	0	0	150.1.20.1:25	192.10.1.50:57568	ESTABLISHED
tcp	0	0	150.1.20.1:25	200.50.2.10:58461	ESTABLISHED
tcp	0	0	150.1.20.1:25	150.10.1.20:60496	ESTABLISHED

Leituras complementares

- Tanenbaum, A. Redes de Computadores (4ª edição), Campus, 2000.
  - Capítulo 5, seção 5.6 (NAT)
  - Capítulo 6, seção 6.1
- Carissimi, A.; Rochol, J; Granville, L.Z; *Redes de Computadores*. Série Livros Didáticos. Bookman 2009.
  - Capítulo 5, seção 5.5.1
  - Capítulo 7, seção 7.2

Gerenciamento de conexões TCP

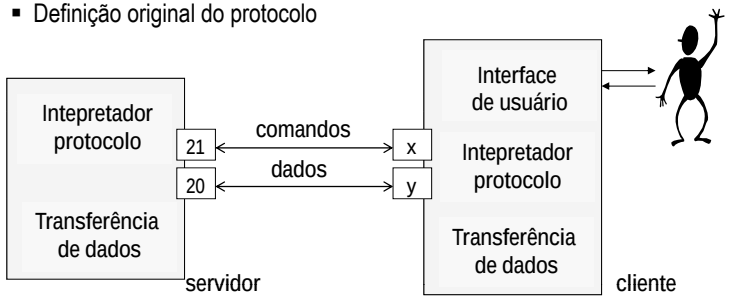
- Fila de requisições
  - Associada a *sockets* TCP no estado listen
    - Processamento da abertura da conexão é realizado pela entidade TCP, e não pelo servidor
    - Requisição de conexão presente na fila ainda não foi aceita pelo servidor
    - Requisição de conexão somente é aceita e removida da fila quando o servidor ativa a função accept
  - Possui capacidade fixa, mas pode ser configurada pelo servidor com a função listen
    - E.g.: *listen(s, 5)*;

TFTP – Trivial File Transfer Protocol

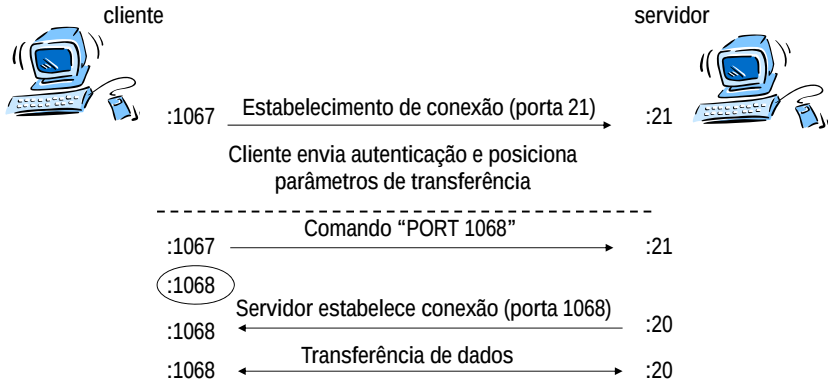
- Similar ao FTP
  - Baseado em UDP (menor em código e mais simples)
  - Conjunto limitado de comandos
  - Não emprega nenhum tipo de autenticação (problema de segurança!!)
- Empregado para:
  - Realizar boot em estações *diskless* (inicialização de máquinas via rede)
  - *Upload* e *download* de *firmware* em dispositivos especificos (embarcados)
- Atende na porta 69 (UDP)

File Transfer Protocol (FTP)

- Utilizado para transferência de arquivos
- Atualmente empregado na forma ftp://url
- Utiliza duas portas (TCP)
  - Porta 20 para dados e porta 21 para comandos
    - Definição original do protocolo

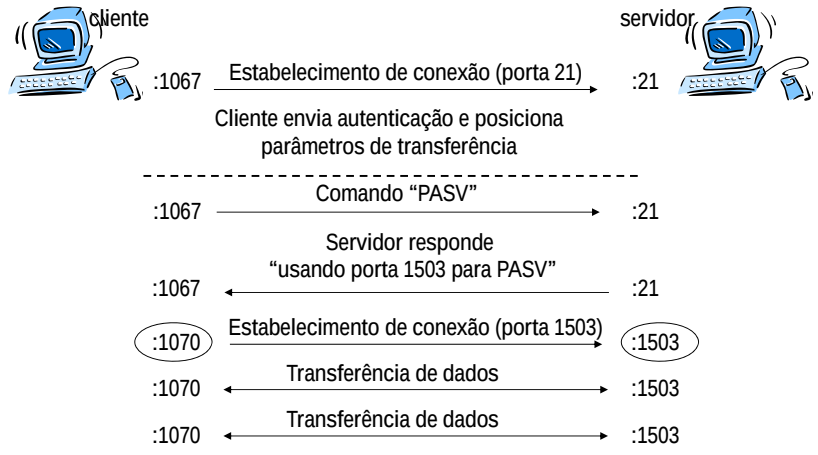


FTP em Modo Ativo



Obs: números de portas acima de 1024 são apenas exemplos

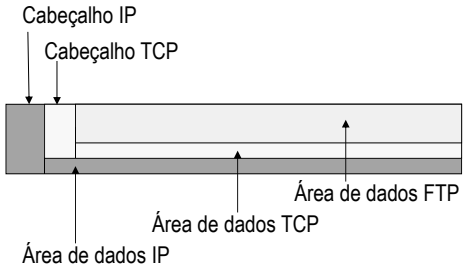
FTP em Modo Passivo



Obs: números de portas acima de 1024 são apenas exemplos

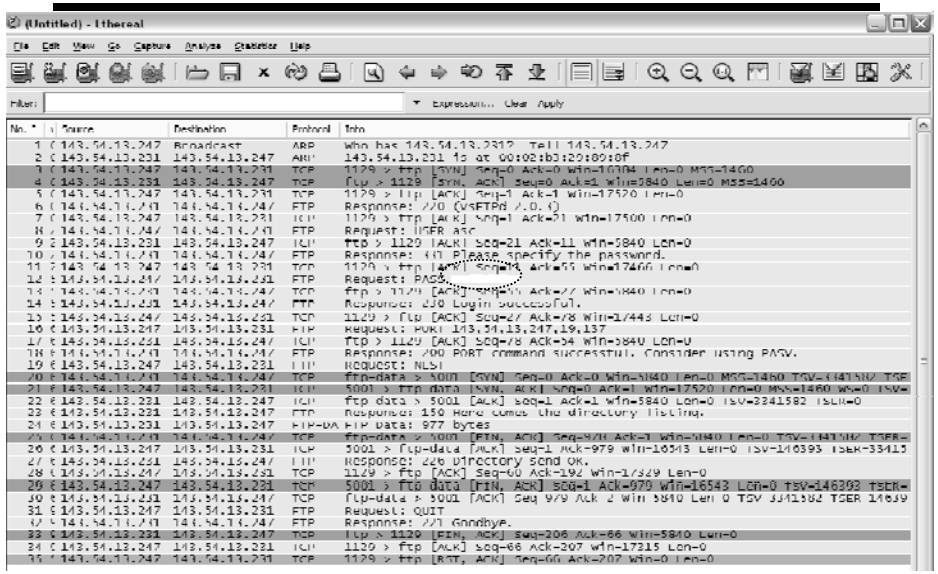
Formato PDU FTP

- Bastante simples: é um *string* delimitado por CR/LF
  - Os strings são interpretados pelos processos envolvidos na sessão FTP

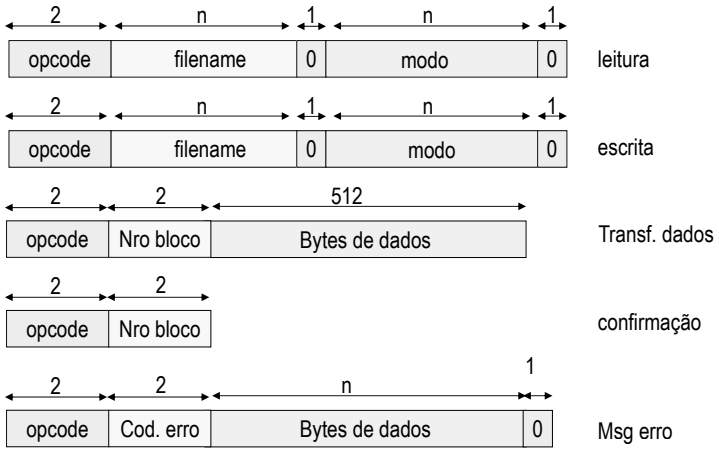


  - Comandos FTP (USER, PASS, PASV, MODE etc)
  - Respostas FTP (códigos de *reply* como 220, 226, 230, etc)

Exemplo: captura de sessão ftp



Formato das PDUs do TFTP



Telnet

- Aplicativo Internet para login remoto
  - Permite um usuário se conectar remotamente em um sistema e enviar comandos como se tivesse "logado" nesse sistema remoto
  - Na verdade, ele é um protocolo de envio de caracteres para um processo remoto.
- Sintaxe: `telnet IP_máquina_remota [porta]`
  - A informação de porta é opcional (default é porta 23/TCP onde atende o servidor telnet)
  - Possível se conectar a outra porta qualquer

    `telnet IP_máquina_remota 25` (envia caracteres → servidor SMTP)

    `telnet IP_máquina_remota 80` (envia caracteres → servidor HTTP)