URI | UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES

30-765

Redes de Computadores II

MSc. Fernando Schubert

CAMADA DE TRANSPORTE - AGENDA

- O serviço transporte;
- Elementos dos protocolos de transporte;
- Protocolo UDP;
- Protocolo TCP;
- Implementação de sockets.

O SERVIÇO DE TRANSPORTE

Tópicos

- Visão geral
- Serviços oferecidos às camadas superiores;
- Primitivasde serviços de transporte.

VISÃO GERAL

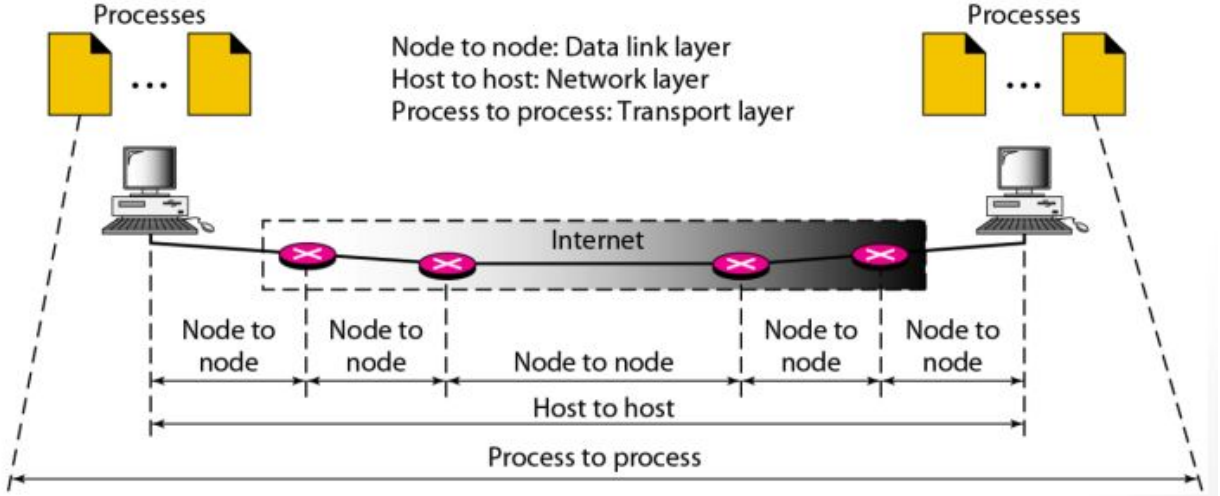

OSI
7 Aplicação
6 Apresentação
5 Sessão
4 Transporte
3 Rede
2 Enlace de dados
1 Física

VISÃO GERAL

- A função básica da Camada de Transporte é aceitar dados da camada de aplicação (camadas superiores), dividi-los em unidades menores em caso de necessidade, passá-los para a Camada de Rede e garantir que todas essas unidades cheguem corretamente à outra extremidade.

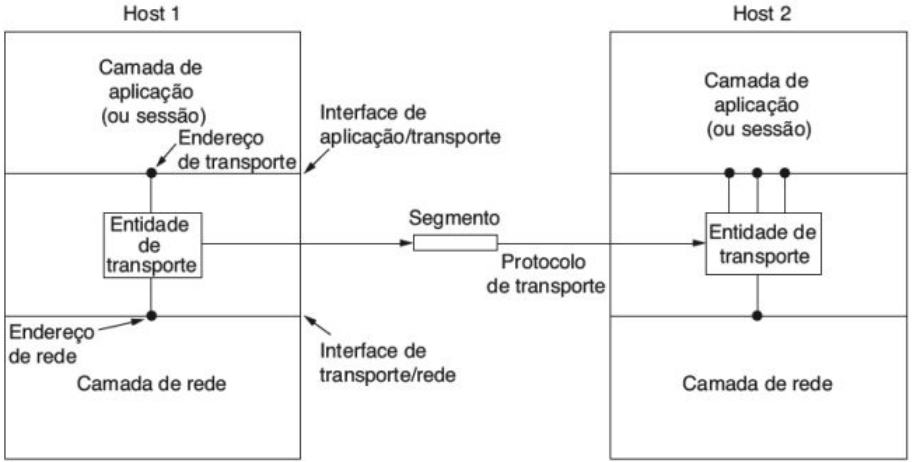
SERVIÇOS OFERECIDOS

- A camada de transporte se baseia na camada de rede para oferecer transporte de dados de um processo em uma máquina de origem para um processo em uma máquina de destino;
- Este transporte de dados deve ser feito com um nível de confiabilidade desejado, independente das redes físicas em uso no momento.



SERVIÇOS OFERECIDOS

- Relacionamento lógico entre as camadas de rede, transporte e aplicação:



- Entidade de transporte: hardware e software que executa o “trabalho”;
  - Segmento: mensagens enviadas entre duas entidades de transporte;
  - Também denominado de TPDU (Transport Protocol Data Unit).

SERVIÇOS OFERECIDOS

- Existem dois tipos de serviços oferecidos:
  - Orientado a conexões;
  - Não orientado a conexões;
- Ambos são semelhantes aos serviços oferecidos pela camada de redes;
- Diante disto, pergunta-se:
  - Por que há duas camadas distintas?
  - Uma única camada não seria suficiente?

SERVIÇOS OFERECIDOS

- O código de transporte funciona inteiramente nas máquinas dos usuários;
  - A camada de rede funciona principalmente nos roteadores, que na maioria dos casos está sob a responsabilidade das concessionárias de comunicação;
- A camada de transporte imuniza as camadas superiores da tecnologia, projeto e imperfeições de rede;
  - Primitivas de serviços podem ser implementadas como chamadas de procedimentos em bibliotecas, tornando-as independentes da rede (primitivas-padrão);
  - Rede (não confiável) vs. Transporte (confiável).

PRIMITIVAS DE SERVIÇOS

- Focaremos neste momento no serviço orientado a conexão;
  - Função da camada de transporte: oferecer um serviço confiável sobre uma rede não confiável;
  - Muitas das aplicações (seus programadores) farão uso da camada de transporte para comunicação, por isso, o serviço de transporte deve ser adequado e fácil de usar;
  - A seguir veremos um conjunto simples de primitivas e um diagrama de estados para ilustrar o funcionamento do serviço da camada de transporte.

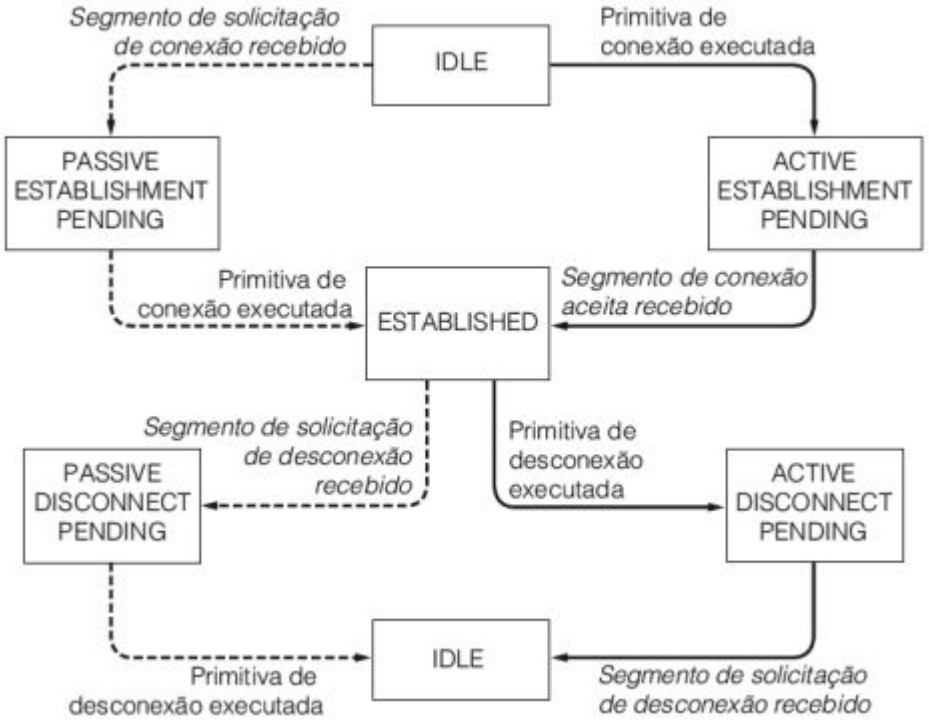
PRIMITIVAS DE SERVIÇOS

- Primitivas para um serviço de transporte simples:

Primitiva	Pacote enviado	Significado
LISTEN	(nenhum)	Bloqueia até algum processo tentar se conectar.
CONNECT	CONNECTION REQ.	Tenta ativamente estabelecer uma conexão.
SEND	DATA	Envia informação.
RECEIVE	(nenhum)	Bloqueia até que um pacote de dados chegue.
DISCONECT	DISCONNECT REQ.	Solicita uma liberação da conexão.

PRIMITIVAS DE SERVIÇOS

- Diagrama de estados:
  - As transições marcadas em itálico são causadas pelos pacotes de chegada;
  - As linhas sólidas mostram a sequência de estados do cliente;
  - As linhas tracejadas mostram a sequência de estados do servidor.



ELEMENTOS DOS PROTOCOLOS DE TRANSPORTE

TÓPICOS

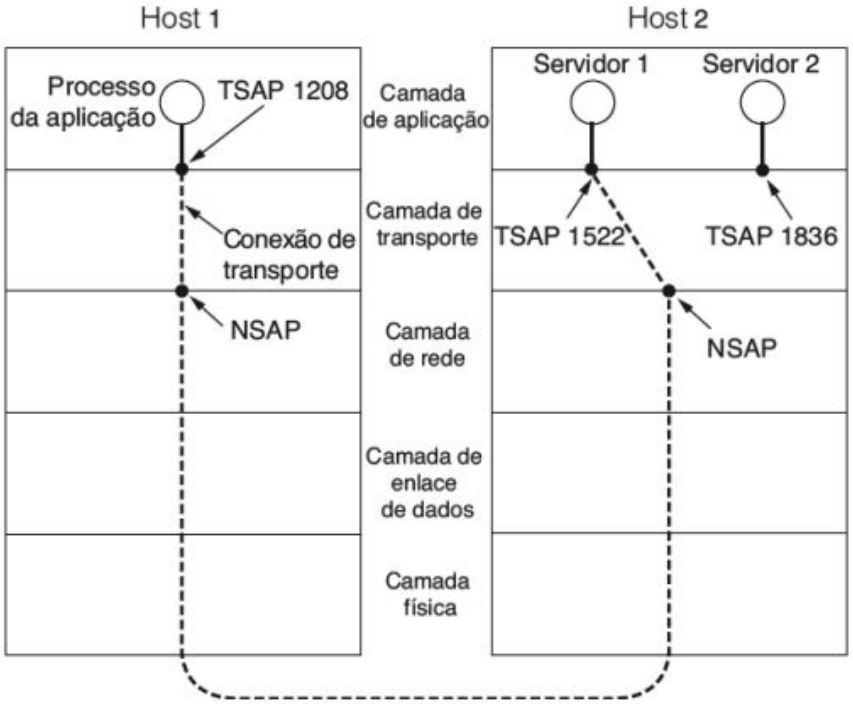
- Endereçamento;
- Portas de Serviços
- Sockets
- Estabelecimento de conexões;
- Encerramento de conexões;
- Controle de erro e fluxo;
- Multiplexação;
- Recuperação de falhas.

ENDEREÇAMENTO

- Para estabelecer uma conexão ou enviar uma mensagem é necessário que um processo da aplicação do cliente saiba como especificar a aplicação remota;
- Na camada de transporte isso é feito a partir de portas, cujo termo genérico é TSAP (Transport Service Access Point);
- Na camada de rede é usado o termo NSAP (Network Service Access Point);

ENDEREÇAMENTO

- Possível Cenário para uma Conexão de Transporte



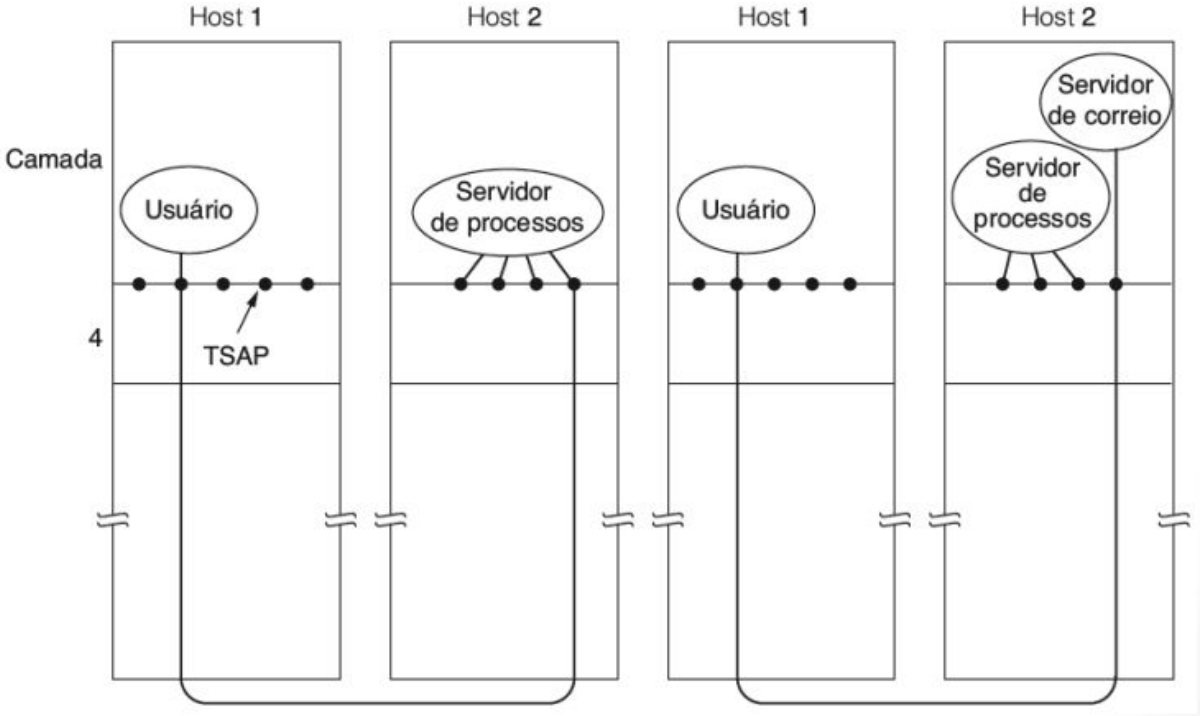
1. Um processo servidor de correio no host 2 se associa ao TSAP 1522;
2. Um processo de aplicação no host 1 transmite uma solicitação CONNECT especificando como origem o TSAP 1208 e como destino o TSAP 1522 do host 2;
3. O processo da aplicação envia a mensagem de correio;
4. O servidor de correio responde que entregará a mensagem;
5. A conexão é encerrada.

ENDEREÇAMENTO

- Como descobrir o endereço TSAP de um servidor remoto?
  - Endereçamento fixo;
  - Portmapper: um processo especial que gerencia o mapeamento de serviços (nome) a portas (número);
  - Ambos possuem endereços estáticos;
- Ocupar portas permanentemente para serviços que normalmente são pouco utilizados é um desperdício;
- Solução: utilizar o protocolo de conexão inicial.

ENDEREÇAMENTO

- **Funcionamento do Protocolo de Conexão Inicial:**



PORTAS DE SERVIÇOS

- • A numeração de portas utiliza 16 bits. Logo, existem 2^16 = 65536 portas a serem utilizadas no total;
- Isso para cada protocolo da camada de transporte. Portanto, 65536 portas para o TCP e 65536 portas para o UDP;
- Conhecer essas portas é fundamental para operar um Firewall de forma satisfatória.

Firewall
Dispositivo de uma rede de computadores que tem por objetivo aplicar políticas de segurança a um determinado ponto da rede. Pode ser do tipo filtro de pacotes, proxy de aplicações, etc.

PORTAS DE SERVIÇOS

- Com tantas portas, como saber todas elas?
- Não precisa conhecer todas, uma vez que a maior parte delas não é especificada;
- Apenas as primeiras 1024 são especificadas;
- Para um administrador de rede, é imprescindível saber pelo menos as portas dos serviços básicos de Rede: Telnet, SSH, FTP, SMTP, POP, HTTP, HTTPS...;
- O uso das portas de 1 a 1024 é padronizado pela IANA (Internet Assigned Numbers Authority);
    - Essa entidade é responsável por alocar portas para determinados serviços;
    - Essas portas são chamadas de well-known ports.

PORTAS DE SERVIÇOS

Serviço	Porta	Protocolo
daytime	13	TCP e UDP
ftp-data	20	TCP
ftp	21	TCP
ssh	22	TCP
telnet	23	TCP
smtp	25	TCP e UDP
name	42	TCP e UDP
nameserver	42	TCP e UDP
tftp	69	TCP e UDP
www	80	TCP
pop3	110	TCP e UDP
netbios-ns	137	TCP e UDP
netbios-dgm	138	TCP e UDP
netbios-ssn	139	TCP e UDP

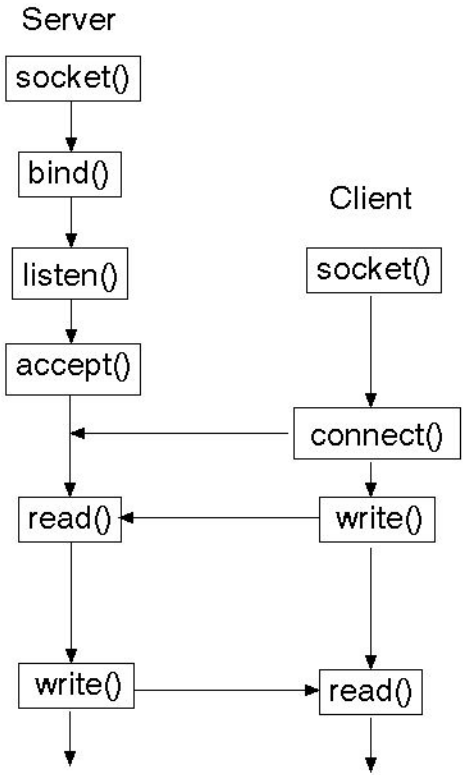
SOCKETS

- Os sockets são diferentes para cada protocolo de transporte. Desta forma, mesmo que um socket TCP possua o mesmo número que um socket UDP, ambos são responsáveis por aplicações diferentes;
- Os sockets de origem e destino são responsáveis pela identificação única da comunicação. Desta forma, é possível a implementação da função conhecida como multiplexação;
- A multiplexação possibilita que haja várias conexões partindo de um único host ou terminando em um mesmo servidor.

SOCKETS

- A formação do socket se dá da seguinte forma:
- Ao iniciar uma comunicação, é especificado para a aplicação o endereço IP de destino e a porta de destino;
- A porta de origem é atribuída dinamicamente pela Camada de Transporte. Ela geralmente é um número aleatório acima de 1024;
- O endereço IP de origem é atribuído pela Camada de Rede.

PRIMITIVAS DE SOCKETS



PRIMITIVAS DE SOCKETS

- `int sockfd = socket(domain, type, protocol)`: criação do *socket*.
  - `sockfd` é o descritor do *socket*;
  - `domain` é o domínio de comunicação (`AF_INET` para IPv4 e `AF_INET6` para IPv6);
  - `type` é o tipo da comunicação (`SOCK_STREAM` para TCP e `SOCK_DGRAM` para UDP);
  - `protocol` é o protocolo usado na comunicação, dependente da implementação (`IPPROTO_TCP` para TCP e `IPPROTO_UDP` para UDP.
- `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`: anexa um endereço local ao *socket*.
  - `*addr` é o endereço do servidor;
  - `addrlen` é o tamanho do endereço do servidor;

PRIMITIVAS DE SOCKETS

- `int listen(int sockfd, int queue_size)`: anuncia o aceite de conexões.
  - `queue_size` é o número máximo de conexões;
- `int new_socket = accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)`: bloqueia até uma tentativa de conexão ser recebida.
- `int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`: conexão feita por um cliente a um servidor.
- `int close(int sockfd)`: encerra a conexão.

PRIMITIVAS DE SOCKETS

- `ssize_t send(int sockfd, const void *buf, size_t len, int flags)`: envia dados através do *socket*.
  - `*buf` são os dados;
  - `len` é o tamanho dos dados;
  - `flags` usado para configuração do envio;
  - Sem `flags` é equivalente ao `write` e é usado para envio de dados no protocolo TCP.
- `ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)`: envia dados através do *socket*, geralmente usado com UDP.
  - `*dest_addr` é o endereço de destino;
  - `addrlen` é o tamanho do endereço de destino;

PRIMITIVAS DE SOCKETS

- `ssize_t recv(int sockfd, void *buf, size_t len, int flags)`: recebe dados através do *socket*.
  - Sem `flags` é equivalente ao `read` e é usado para envio de dados no protocolo TCP.
- `ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)`: recebe dados através do *socket*, geralmente usado com UDP.
  - `*src_addr` é o endereço de origem;
  - `*addrlen` é o endereço de uma variável que guarda o tamanho do endereço de origem;

SOCKETS - ATIVIDADE

1. Sockets simples: crie um cliente e servidor em C/C++ que faça o seguinte:
   a. O servidor faz o bind na porta 7777
   b. O servidor espera por um número do cliente, imprime localmente se o número é par ou ímpar juntamente com o IP de origem
   c. Retorna ao cliente uma mensagem de erro se o valor é inválido
   d. Se for válido retorna se o valor é par ou ímpar
   e. O cliente por sua vez envia ao servidor um valor e espera a resposta.

SOCKETS - ATIVIDADE

2. Sockets complexo:
   - A partir do exemplo implementado anteriormente:
     - Adicione suporte a múltiplas threads para atender vários clientes ao mesmo tempo, ou seja, cada nova conexão de cliente gera uma nova thread.

SOCKETS - ATIVIDADE

3. Transferencia de arquivos utilizando sockets:
    - Servidor de Arquivos:
        - Crie um cliente em C que consiga enviar um arquivo de até 1MB para o servidor e rejeite arquivos maiores
        - Crie um servidor que receba o arquivo na porta 9999 e salve no sistema de arquivos local.
        - Permita conexões simultâneas de vários clientes no servidor
        - Gerencie conflitos de nomes dos arquivos, por ex, se 2 ou mais clientes enviar o mesmo arquivo ele deve ser salvo em locais diferentes e retornada uma URL com o caminho para o arquivo ser acessado via outro servidor (Web)
    - Servidor web:
        - Utilize uma solução existente para o servidor web, Apache, nginx, IIS
        - O servidor deve mapear o local onde os arquivos salvos pelo servidor de arquivos estão armazenados
        - Permita que os clientes possam utilizar seus navegadores para acessar os arquivos enviados (priorizar formatos que são aceitos pelo servidor web: html, imagens)
        - O cliente deve utilizar a URL informada pelo servidor de arquivos para acessar o arquivo.

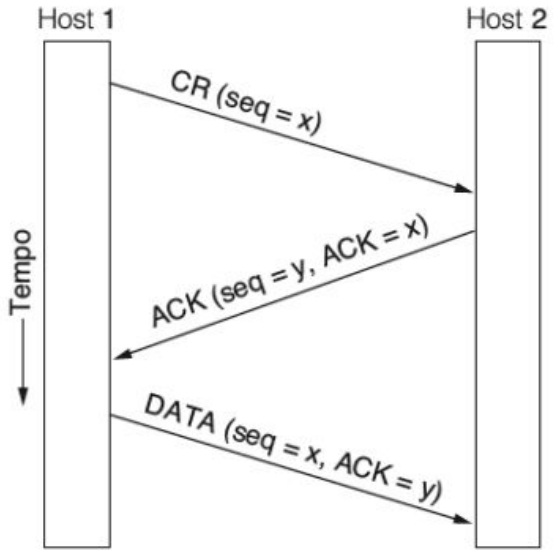
    - Dockerize os dois servidores e exponha as suas portas no sistema principal para serem acessíveis pela rede

ESTABELECIMENTO DE CONEXÕES

- Estabelecer uma conexão pode parecer simples, mas não é;
- Pacotes podem ser perdidos, atrasados, corrompidos e duplicados;
- Uma solução foi proposta por Tomlinson em 1975: o handshake de três vias;
  - Cada segmento é numerado;
  - Esta numeração não se repete por um tempo T;
  - Esquema a seguir.

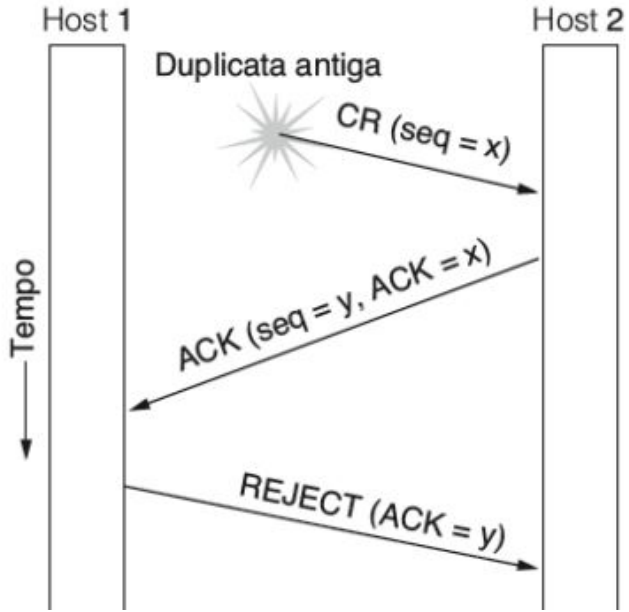
ESTABELECIMENTO DE CONEXÕES

- **Handshake de Três Vias (1/3):**
  - Situação normal:



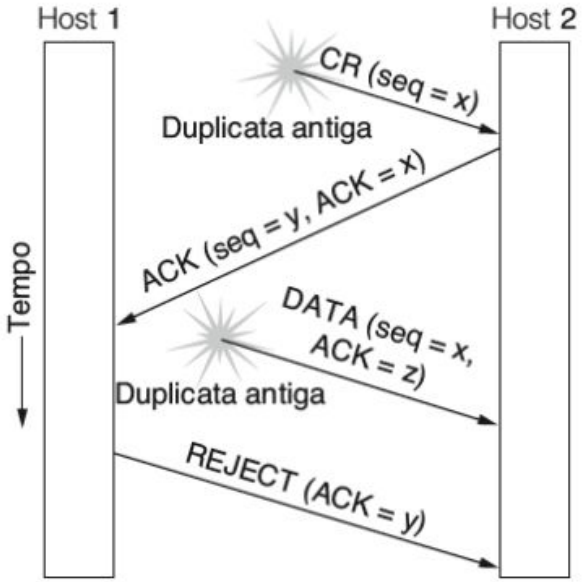
ESTABELECIMENTO DE CONEXÕES

- **Handshake de Três Vias (2/3):**
  - Duplicata antiga de CONNECTION REQUEST que surge repetidamente:



ESTABELECIMENTO DE CONEXÕES

- **Handshake de Três Vias (3/3):**
  - CONNECTION REQUEST e ACK duplicadas:



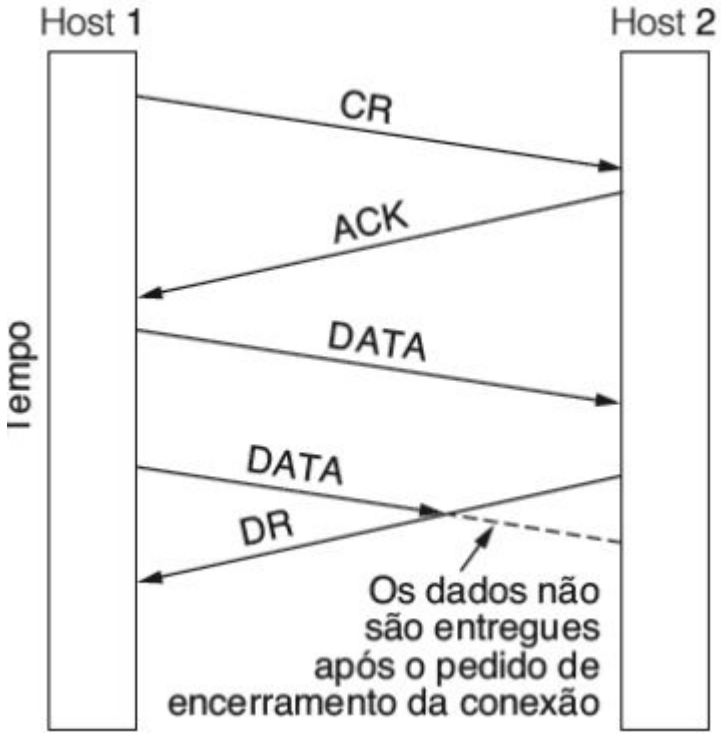
ENCERRAMENTO DE CONEXÕES

Pode acontecer de duas formas:

- **Assimétrico:**
  - Trata a conexão de maneira semelhante ao sistema telefônico;
  - Quando um dos interlocutores termina, a conexão é interrompida;
- **Simétrico:**
  - Trata a conexão como duas conexões unidimensionais isoladas;
  - Exige que cada conexão seja encerrada separadamente.

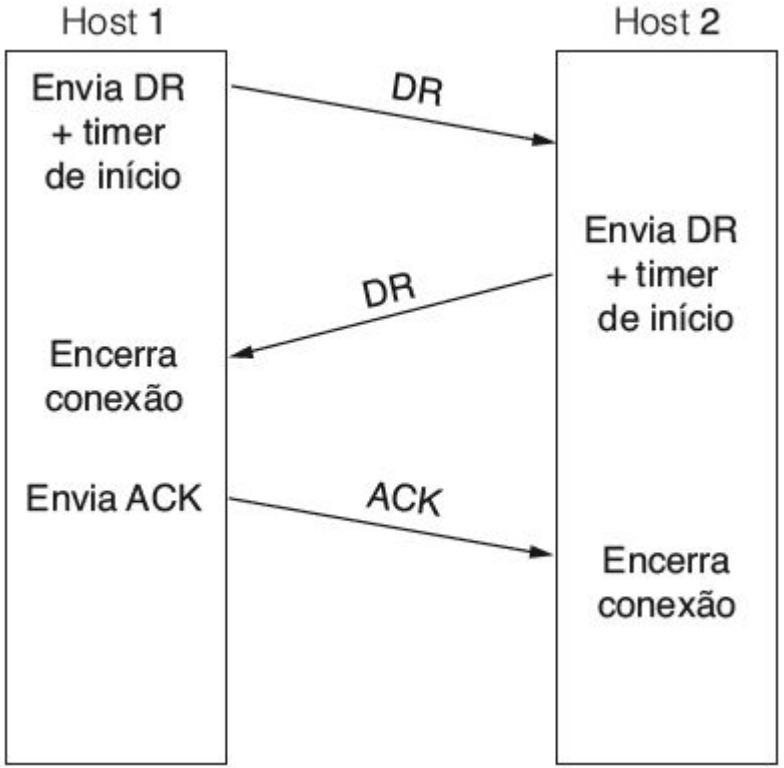
ENCERRAMENTO DE CONEXÕES

- Encerramento Assimétrico:
  a. Dados podem ser perdidos:



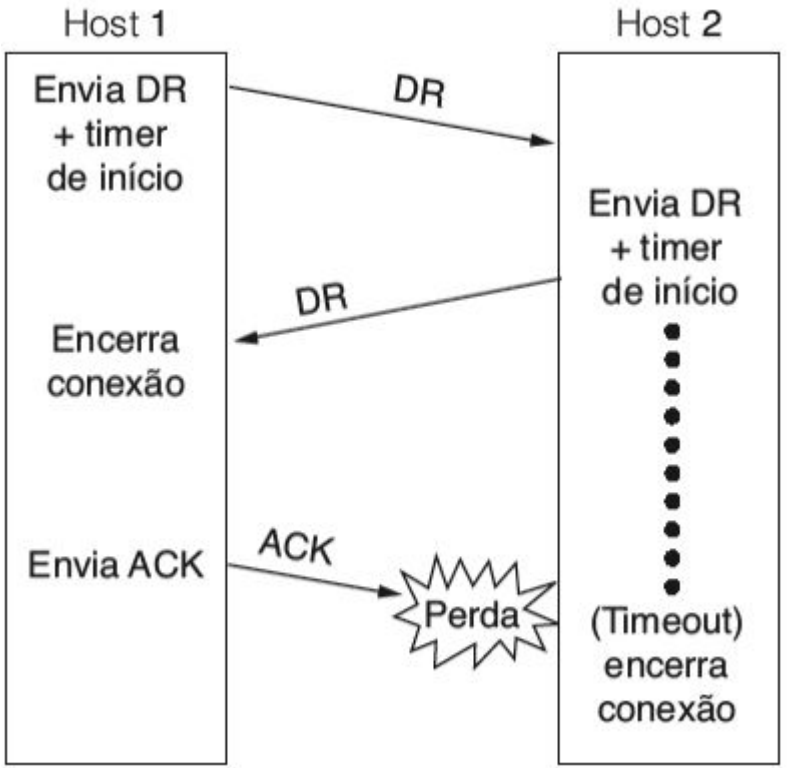
ENCERRAMENTO DE CONEXÕES

- Encerramento Simétrico (1/5):
  a. Indicado quando uma quantidade fixa de dados será transmitida e é possível saber quando a transmissão termina;
- Situação 1: Caso normal de handshake de três vias:



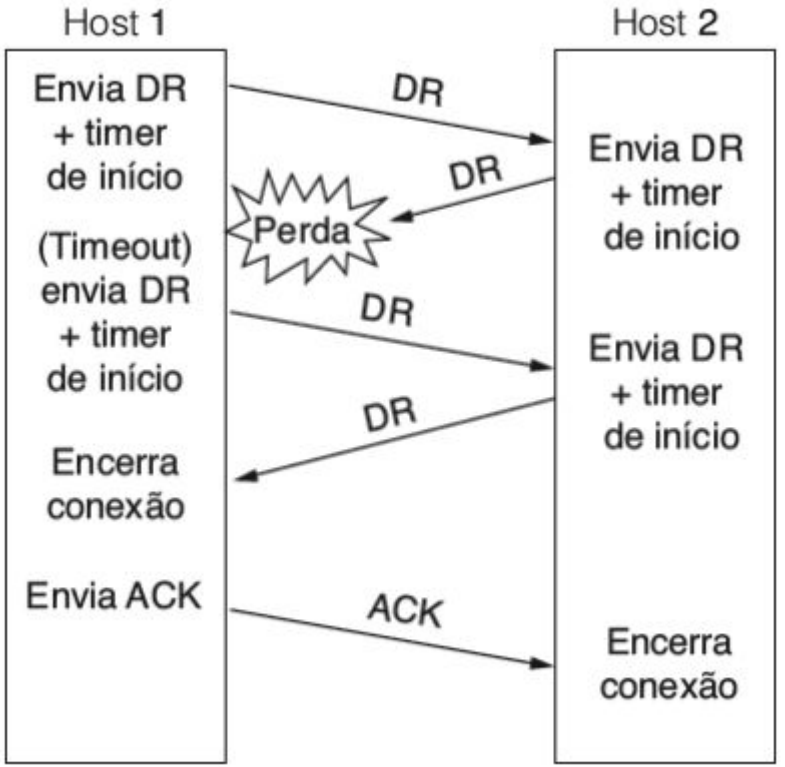
ENCERRAMENTO DE CONEXÕES

- Encerramento Simétrico (2/5):
  a. Situação 2: ACK final perdido:



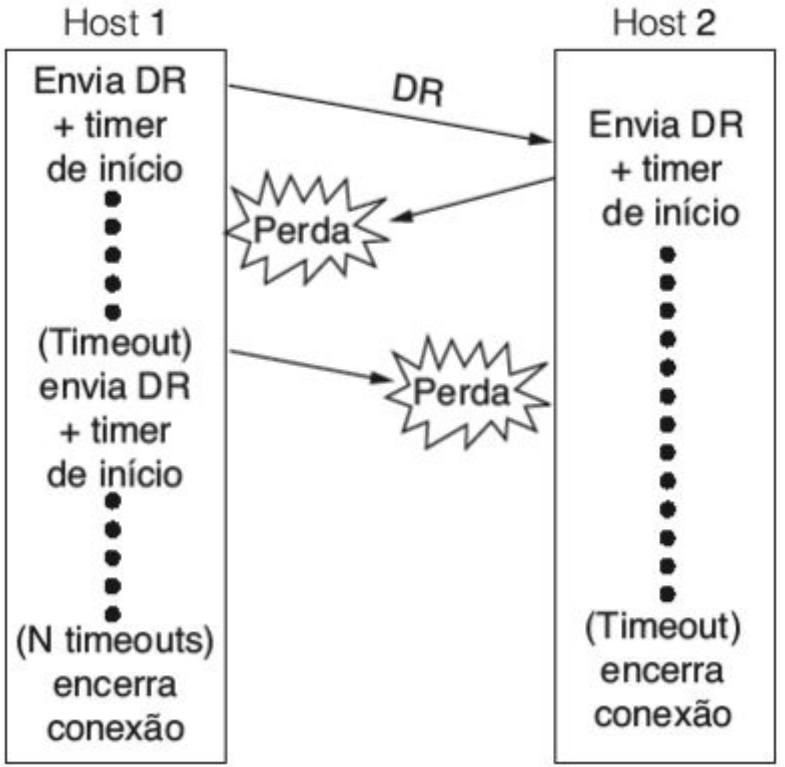
ENCERRAMENTO DE CONEXÕES

- Encerramento Simétrico (3/5):
  a. Situação 3: Resposta perdida:



ENCERRAMENTO DE CONEXÕES

- Encerramento Simétrico (4/5):
  a. Situação 4: Resposta perdida e DRs subsequentes perdidos:



ENCERRAMENTO DE CONEXÕES

Encerramento Simétrico (5/5):

- Ainda assim existe a possibilidade de falha;
- Imagine que a primeira DR e todas as demais retransmissões se perdessem;
- Um host encerraria a conexão e o outro não, ocasionando em uma conexão “semiaberta”;
- Uma alternativa é que a conexão seja encerrada após um determinado tempo de inatividade.

CONTROLE DE ERRO E FLUXO

- Estabelecido como as conexões podem ser iniciadas e finalizadas, é hora de verificar algumas questões relacionadas ao gerenciamento das conexões enquanto em uso;
- As principais questões são:
    - Controle de erro: garantir que os dados sejam entregues em um nível de confiabilidade desejado, normalmente que todos os dados sejam entregues sem nenhum erro;
    - Controle de fluxo: impedir que um transmissor rápido sobrecarregue um receptor lento;
- Mas isso já não foi feito na camada de enlace?

CONTROLE DE ERRO E FLUXO

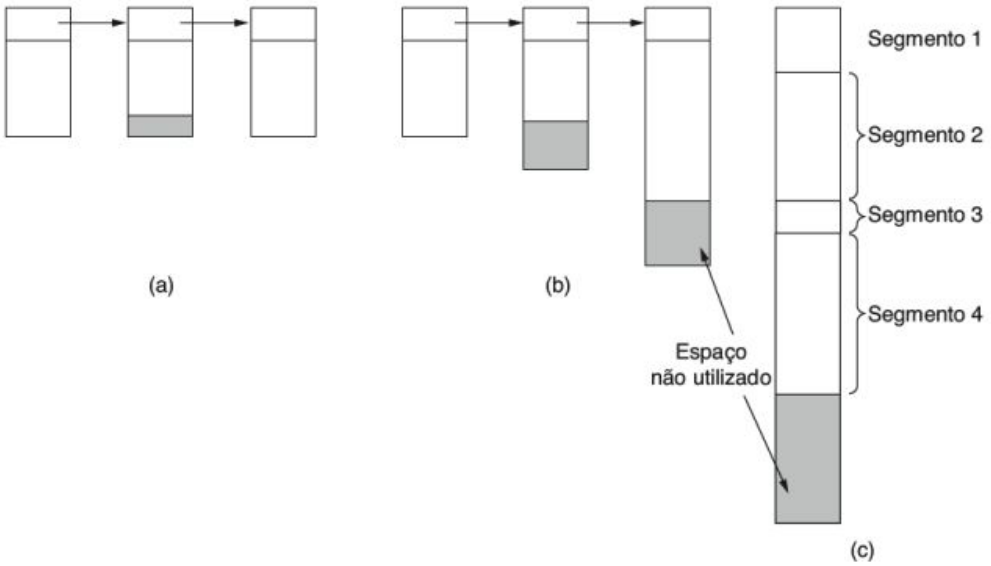
- Sim, e na camada de transporte utiliza-se dos mesmos mecanismos estudados na camada de enlace;
- Embora utilize os mesmos mecanismos, existem diferenças em função e grau;
- Controle de erro:
  - O checksum da camada de enlace protege um quadro quando ele atravessa um único enlace;
  - O checksum da camada de transporte protege um segmento enquanto ele atravessa um caminho de rede inteiro;
  - Erros de quadros podem passar despercebidos e serem identificados apenas na camada de transporte;
    - Ex.: erro de processamento em um roteador;

CONTROLE DE ERRO E FLUXO

- Controle de Fluxo (1/4):
  - Feito através da gerência de buffer e do tamanho da janela deslizante;
  - Reserva de espaço em buffer de um receptor limita o fluxo máximo de transmissão de um transmissor;
    - Durante a conexão, são feitas negociações de reserva de buffer em um receptor para uma conexão específica;
    - O transmissor deve possuir buffer para armazenar segmentos que ainda não foram confirmados;
    - Como definir o tamanho dos buffers?

CONTROLE DE ERRO E FLUXO

- Controle de Fluxo (2/4):
  - Tamanho dos *buffers*:



(a) Encadeamento de *buffers* de tamanho fixo.
(b) Encadeamento com tamanho variável. (c) Um grande *buffer* circular por conexão.

CONTROLE DE ERRO E FLUXO

- Controle de Fluxo (3/4):

- Alocação dinâmica de *buffer*:

	A	Mensagem	B	Comentários
1	→	<request 8 buffers>	→	A deseja 8 buffers
2	←	<ack = 15, buf = 4>	←	B concede mensagens 0-3 apenas
3	→	<seq = 0, data = m0>	→	A tem 3 buffers restantes agora
4	→	<seq = 1, data = m1>	→	A tem 2 buffers restantes agora
5	→	<seq = 2, data = m2>	•••	Mens. perdida, mas A acha que tem 1 restante
6	←	<ack = 1, buf = 3>	←	B confirma 0 e 1, permite 2-4
7	→	<seq = 3, data = m3>	→	A tem 1 buffer restante
8	→	<seq = 4, data = m4>	→	A tem 0 buffers restantes e deve parar
9	→	<seq = 2, data = m2>	→	A atinge o timeout e retransmite
10	←	<ack = 4, buf = 0>	←	Tudo confirmado, mas A ainda bloqueado
11	←	<ack = 4, buf = 1>	←	A pode agora enviar 5
12	←	<ack = 4, buf = 2>	←	B encontrou novo buffer em algum lugar
13	→	<seq = 5, data = m5>	→	A tem 1 buffer restante
14	→	<seq = 6, data = m6>	→	A agora está bloqueado novamente
15	←	<ack = 6, data = m6>	←	A ainda está bloqueado
16	•••	<ack = 6, buf = 4>	←	Impasse em potencial

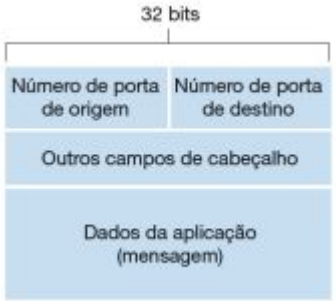
As setas mostram a direção de transmissão. (...) indica perda segmentos.

CONTROLE DE ERRO E FLUXO

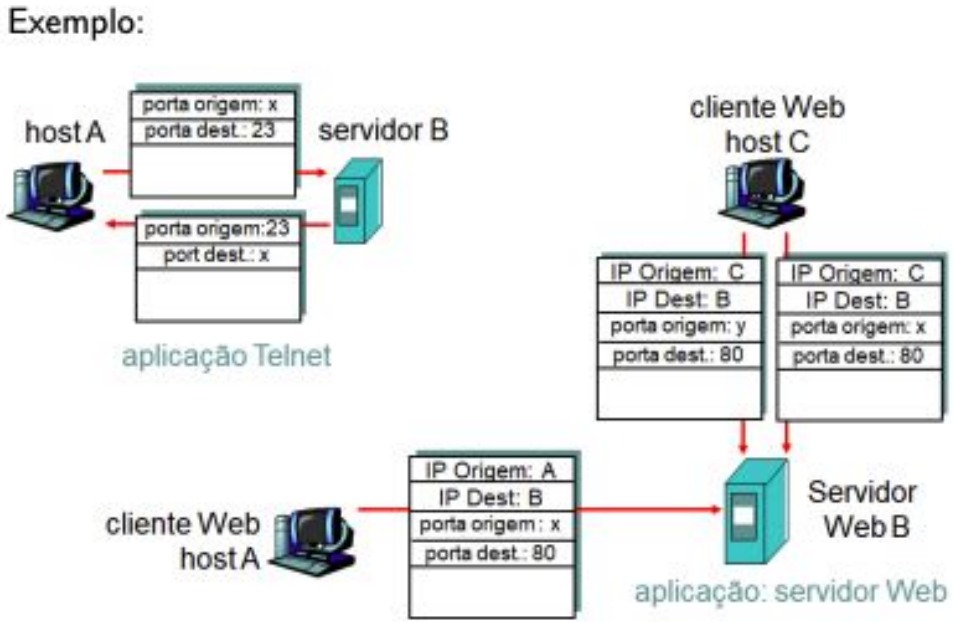
- Controle de Fluxo (4/4):
  - Quando o espaço em buffer deixar de limitar o fluxo máximo, surgirá outro gargalo: a capacidade de transporte da rede;
  - Então, é necessário definir um mecanismo que limite as transmissões com base nesta capacidade;
  - Uma solução proposta por Belsnes é o ajuste dinâmico do tamanho da janela deslizante:
    - Se a rede puder transmitir c segmentos/s e o tempo de ciclo for de r (considerando todos os fatores de transmissão e confirmação), então o tamanho da janela deverá ser de c*r;
    - Como a capacidade da rede varia com o tempo, o tamanho da janela deverá ser ajustado com frequência;
  - Esta estratégia permite fazer o controle de fluxo e de congestionamento através do tamanho da janela deslizante;

MULTIPLEXAÇÃO

- Multiplexação/demultiplexação: baseada em número de porta do transmissor, número de porta do receptor e endereços IP:
  - Números de porta origem e destino em cada segmento;
  - Lembre: portas com números bem conhecidos são usadas para aplicações específicas.



MULTIPLEXAÇÃO

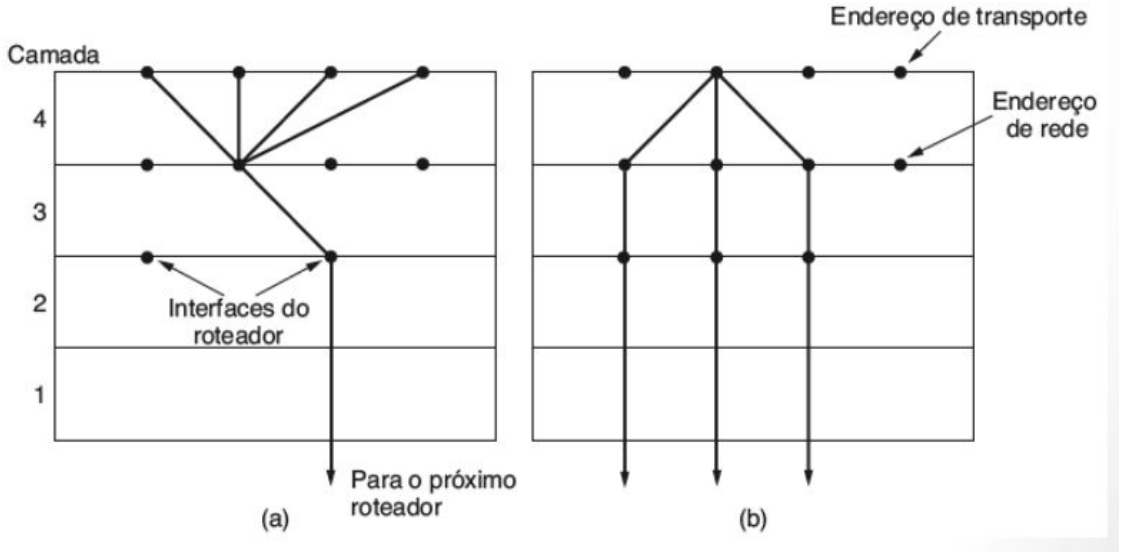


MULTIPLEXAÇÃO

- Muitos hosts são multiprogramados;
- Isso significa que muitas conexões estarão entrando e saindo de cada host, e processos executam de forma concorrente.
- É responsabilidade da Camada de Transporte multiplexar todas as comunicações para a Camada de Rede e posteriormente determinar a qual processo uma mensagem pertence;

MULTIPLEXAÇÃO

- Pode ser de diversas formas:
  - a) Caso exista apenas um endereço de rede;
  - b) Um usuário que necessitar de mais largura de banda pode utilizar vários caminhos de rede (multiplexação inversa).



RECUPERAÇÃO DE FALHAS

- Poderão ocorrer falhas nos hosts ou nos roteadores;
- Com a entidade de transporte totalmente no host, uma falha em roteador é facilmente tratada:
  - As entidades de transporte esperam segmentos perdidos o tempo todo e sabem como lidar com eles usando retransmissões;
- O problema maior é quando um host falha:
  - Em particular, pode-se desejar que o cliente continue funcionando quando um servidor falhar e retornar instantes depois;
  - Sempre haverá situações em que o protocolo não recuperará o funcionamento de modo apropriado.

RECUPERAÇÃO DE FALHAS

- Diferentes combinações de estratégias de cliente e servidor:

	Estratégia usada pelo host receptor					
	Primeiro ACK, depois gravar			Primeiro gravar, depois ACK		
Estratégia usada pelo host transmissor	AC(W)	AWC	C(AW)	C(WA)	WAC	WC(A)
Sempre retransmitir	OK	DUP	OK	OK	DUP	DUP
Nunca retransmitir	LOST	OK	LOST	LOST	OK	OK
Retransmitir em S0	OK	DUP	LOST	LOST	DUP	OK
Retransmitir em S1	LOST	OK	OK	OK	OK	DUP

OK = Protocolo funciona corretamente
DUP = Protocolo gera uma mensagem duplicada
LOST = Protocolo perde uma mensagem

S0: nenhum seguimento pendente; **S1**: um seguimento pendente;

A: enviar confirmação (ACK); **W**: gravar no processo de saída; **C**: sofrer uma pane;

- **Conclusão**: Recuperação de falha na camada **N** só pode ser realizada na camada **N+1**.

PROTOCOLO UDP

TÓPICOS

- Diferentes combinações de estratégias de cliente e servidor:

Estratégia usada pelo host transmissor	Estratégia usada pelo host receptor: Primeiro ACK, depois gravar			Primeiro gravar, depois ACK		
	AC(W)	AWC	C(AW)	C(WA)	WAC	WC(A)
Sempre retransmitir	OK	DUP	OK	OK	DUP	DUP
Nunca retransmitir	LOST	OK	LOST	LOST	OK	OK
Retransmitir em S0	OK	DUP	LOST	LOST	DUP	OK
Retransmitir em S1	LOST	OK	OK	OK	OK	DUP

OK = Protocolo funciona corretamente
DUP = Protocolo gera uma mensagem duplicada
LOST = Protocolo perde uma mensagem

S0: nenhum seguimento pendente; **S1**: um seguimento pendente;

A: enviar confirmação (ACK); **W**: gravar no processo de saída; **C**: sofrer uma pane;

- **Conclusão**: Recuperação de falha na camada **N** só pode ser realizada na camada **N+1**.