



# Camada de Transporte do RM-OSI

Ana Luiza Teodoro Cupertino - [analuizatcupertino@gmail.com](mailto:analuizatcupertino@gmail.com)

Brenner de Souza Borges - [brenner.borges@hotmail.com](mailto:brenner.borges@hotmail.com)

Diogo Mafra Q. B. Magalhães - [diogomqbm13@gmail.com](mailto:diogomqbm13@gmail.com)

Felipe Costa Mendes - [felipe0211@hotmail.com](mailto:felipe0211@hotmail.com)

Thiago Lucas Carvalho - [thiagol.carvalho97@gmail.com](mailto:thiagol.carvalho97@gmail.com)

# Sumário



- **6 - A camada de transporte**
  - 6.1 - Problemas no design da camada de transporte
  - 6.1.1 - Serviços providos para a camada de sessão
  - 6.1.2 - Qualidade do serviço
  - 6.1.3 - As primitivas do serviço de transporte OSI
  - 6.1.4 - Protocolos de transporte
  - 6.1.5 - Elementos dos protocolos de transporte
- **6.2 - Gerenciamento de Conexão**
  - 6.2.1 - Endereçamento
  - 6.2.2 - Estabelecendo conexão
  - 6.2.3 - Terminando conexão
  - 6.2.4. Gerenciamento de Conexão Baseado em Timer
  - 6.2.5. Controle de Fluxo e Buffering
  - 6.2.6. Multiplexação
  - 6.2.7. Crash Recovery
- **6.3 Um simples protocolo de transporte sob o X.25**
  - 6.3.1. Exemplo do Serviço de Primitivas
  - 6.3.2. Exemplo da Entidade de Transporte
- **6.4 Exemplos da Camada de Transporte**
  - 6.4.1. A Camada de Transporte em Redes Públicas
  - 6.4.2. A Camada de Transporte na ARPANET (TCP)
  - 6.4.3. A Camada de Transporte no MAP e TOP
  - 6.4.4. Camada de Transporte no USENET



## 6 - A camada de transporte

- A camada de transporte é a quarta camada do modelo OSI e é o coração de toda a hierarquia de protocolos.
- Sua função é promover uma transferência de dados de forma confiável e econômica (ou seja de forma que não se consuma muitos recursos para tal controle) de uma máquina para outra, independente das redes físicas que estejam sendo usadas.




## 6.1 - Problemas no design da camada de transporte


- Nesta seção é apresentado alguns dos problemas que desenvolvedores têm ao fazer o design da camada de transporte.
- Esses problemas incluem o tipo e a qualidade do serviço provido para a camada de transporte, e as primitivas desta providas para invocar o serviço.
- Além disso, temos também uma discussão inicial dos protocolos necessários para realizar o serviço de transporte.




### **6.1.1 - Serviços providos para a camada de sessão**

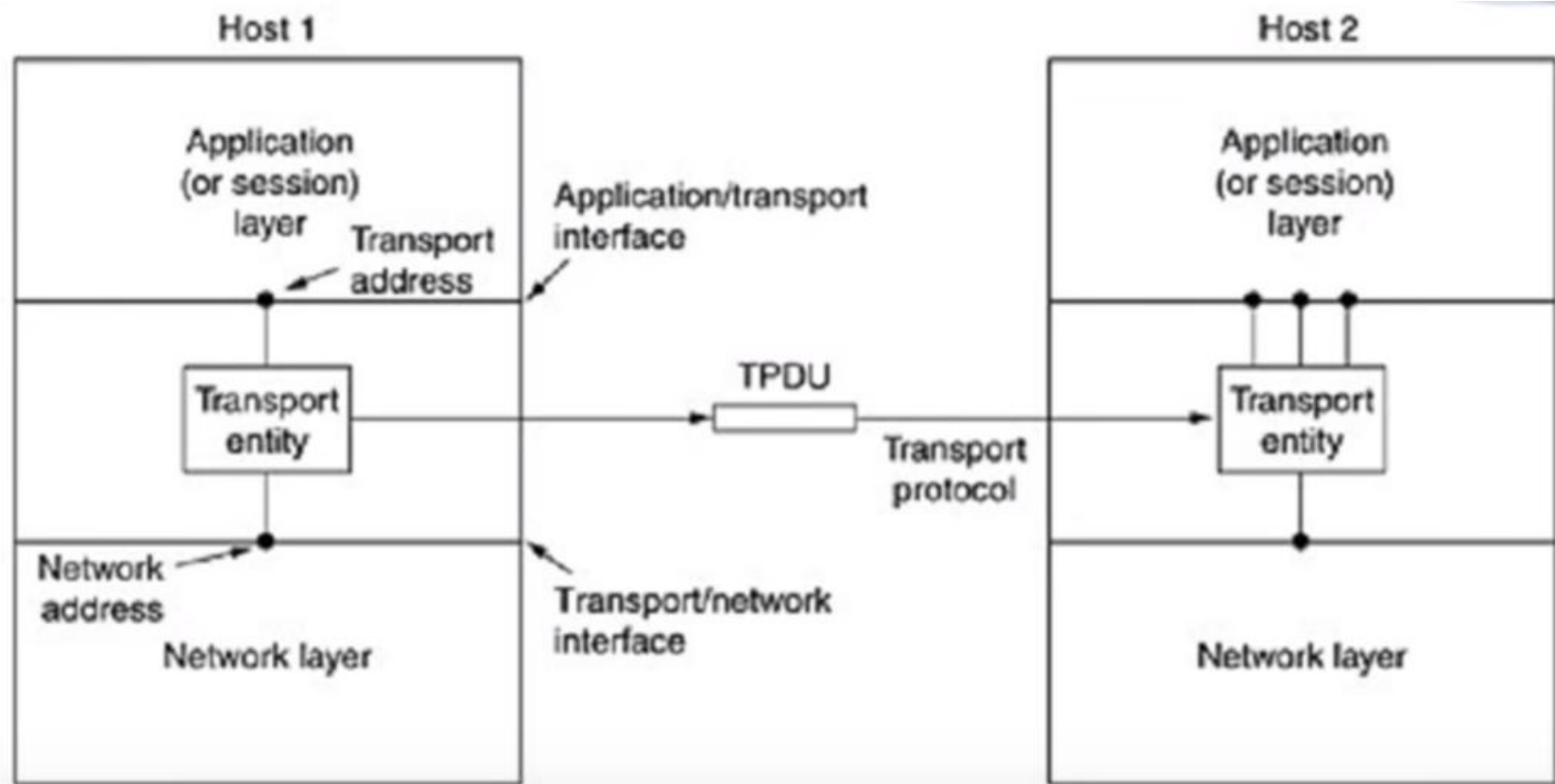
- Para que a camada de transporte faça sua função (transferência de dados de forma confiável e econômica), é necessário o uso dos serviços oferecidos pela camada de rede.
- O hardware/software da camada de transporte que faz o trabalho é chamado de entidade de transporte.

- 
- Existem dois tipos de serviço de transporte: orientado à conexão e sem conexão.
  - No transporte orientado à conexão, existem três fases nas conexões: estabelecimento, transferência de dados e encerramento.
  - Essas características acima são similares com as da camada de rede, porém a camada de transporte é essencial pois possibilita um serviço de transporte mais confiável que o serviço de rede.
  - Pacotes perdidos, dados deformados e até N-RESETs de rede podem ser detectados e compensados pela camada de rede.

- 
- As primitivas do serviço de transporte podem ser desenvolvidas para serem independentes das primitivas da camada de rede, dependendo do tipo de rede.
  - A camada de transporte permite que programas sejam escritos com um conjunto de primitivas e estes podem funcionar em uma grande variedade de redes, sem ter que se preocupar com transmissões não confiáveis e diferentes interfaces de sub-redes.

- 
- Por conta de problemas de confiabilidade de redes e divergência de primitivas, é feito uma distinção entre as camadas 1 até 4, e 5 até 7. Sendo as camadas mais em baixo as provedoras de serviço de transporte, e as últimas as usuárias do serviço de transporte.
  - Esta distinção tem um impacto considerável no design das camadas, e coloca a camada de transporte em uma posição chave, já que ela faz o limite entre os provedores e usuários do serviço de transmissão confiável de dados.











## 6.1.2 - Qualidade do serviço


- Uma outra maneira de olharmos para a camada de transporte é considerando a sua função primária de melhorar a qualidade de serviço (QOS - Quality of Service) provida pela camada de rede.
- A QOS é definida por um número específico de parâmetros.
- O serviço de transporte OSI permite que o usuário especifique valores preferidos, aceitáveis e inaceitáveis para os parâmetros quando uma conexão é iniciada.

- 
- É de responsabilidade da camada de transporte examinar estes parâmetros e dependendo dos serviços de rede disponíveis, determinar se poderá ou não prover o determinado serviço.
  - As QOS são: atraso de estabelecimento de conexão, probabilidade de falha do estabelecimento de conexão, taxa de transferência, atraso de trânsito, taxa de erro residual, probabilidade de falha de transferência, atraso de encerramento da conexão, probabilidade de falha do encerramento da conexão, proteção, prioridade e resiliência.

- 
- **Atraso de estabelecimento de conexão:** é a quantidade de tempo decorrente entre a requisição de uma conexão de transporte e a confirmação recebida pelo usuário do serviço de transporte.
  - **Probabilidade de falha do estabelecimento de conexão:** é a chance da conexão não ser estabelecida até o tempo máximo de atraso permitido (pode ocorrer devido a congestionamento de conexão, falta de espaço ou problemas internos).
  - **Taxa de transferência:** mede o número de bytes de dados do usuário transferidos por segundo, medidos em um intervalo de tempo recente.

- 
- **Atraso de trânsito:** mede o tempo entre a mensagem ter sido enviada pelo usuário de transporte na máquina de origem e o recebimento desta pelo usuário de transporte da máquina de destino.
  - **Taxa de erro residual:** mede o número de mensagens perdidas ou ilegíveis, como uma fração do total de mensagens enviadas num período de amostragem.
  - **Probabilidade de falha de transferência:** mede quão bem o serviço de transporte está sendo realizado.

- 
- **Atraso de encerramento da conexão:** é a quantidade de tempo decorrente entre a o usuário de transporte iniciar o encerramento da conexão e a conexão ser realmente encerrada na outra máquina.
  - **Probabilidade de falha do encerramento da conexão:** é a fração das tentativas de encerramento da conexão que não foram concluídas dentro do intervalo acordado de atraso de encerramento.
  - **Proteção:** provém um meio para o usuário de transporte dizer se quer que a camada de transporte provenha proteção contra terceiras partes não autorizadas, poderem ler ou modificar o dado transmitido.

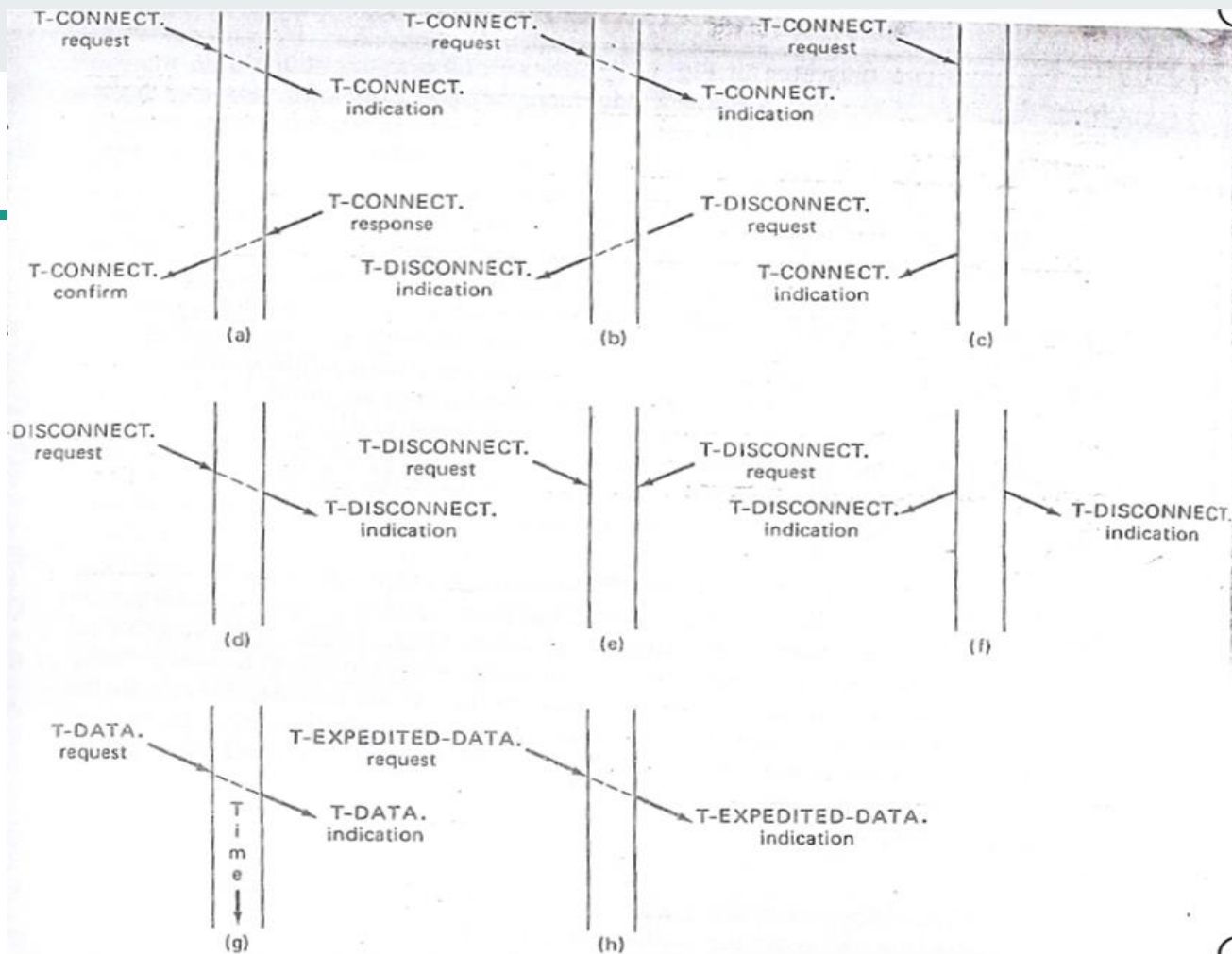
- 
- **Prioridade:** provém um meio para o usuário de transporte indicar quais das suas conexões são as mais importantes, e caso haja congestionamento, elas sejam priorizadas.
  - **Resiliência:** dá a probabilidade da camada de transporte encerrar uma conexão espontaneamente devido a problemas internos ou congestionamento.
  - A **opção de negociação** é a definição de parâmetros aceitáveis para o estabelecimento de uma conexão, e estas definições duram até o final da conexão. As definições do serviço de transporte OSI (ISO 8027) não dão a codificação ou valores para os parâmetros da QOS.





### 6.1.3 - As primitivas do serviço de transporte OSI

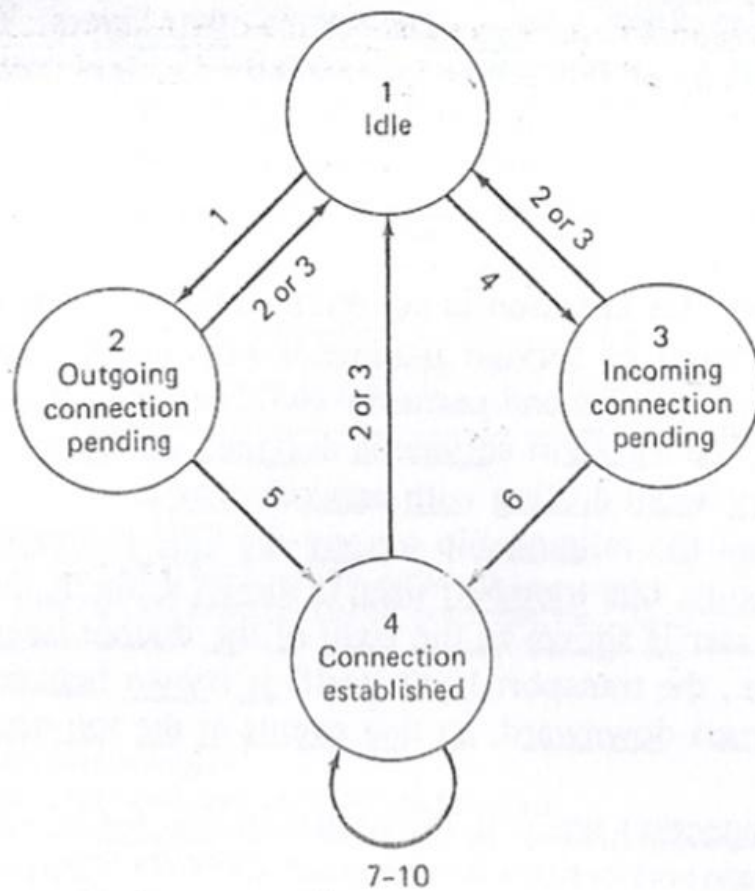
- As primitivas do serviço de transporte OSI são providas tanto para os serviços orientados à conexão, quanto para os sem conexão.
- O serviço de transporte foi desenvolvido para aliviar os programas que o usam de ter que se preocupar em lidar com erros de rede.





- 
- Provedores de transporte bem desenvolvidos normalmente não emitem indicativos de primitivas T-DISCONNECT espontâneos, mas há circunstâncias que não existem outros caminhos a serem tomados.
  - A primitiva T-EXPEDITED-DATA pode ser usada para enviar dados que passam na frente de outros dados na fila. Essa primitiva normalmente é usada para transmitir as chaves: BREAK, DEL ou interrupt key, que podem ser digitadas no terminal para interromper o programa.
  - Existem regras para estabelecer a ordem em que as primitivas de transporte são usadas.


- 
- IDLE: nenhuma conexão foi estabelecida e não houve tentativas para estabelecer uma. Tanto conexões de entrada e saída são possíveis.
  - CONEXÃO DE SAÍDA PENDENTE: uma requisição T-CONNETC foi feita, a resposta do outro par não foi recebida ainda.
  - CONEXÃO DE ENTRADA PENDENTE: uma requisição T-CONNETC foi recebida, ainda não foi aceita ou rejeitada.
  - CONEXÃO ESTABELECIDADA: uma conexão válida foi estabelecida e a transferência de dados pode ser iniciada.






## 6.1.4 - Protocolos de transporte

- O serviço de transporte é implementado por um protocolo de transporte usado entre duas entidades de transporte. Ele tem que lidar com controle de erros, controle de fluxo, e outros problemas.
- Os tipos de serviço de rede estão separados em três categorias:
- A: Sem falhas, serviço livre de erros e sem N-RESETS;
- B: Entrega perfeita de pacotes, mas com N-RESETS;
- C: Serviço não confiável, com pacotes perdidos e duplicados, e possíveis N-RESETS;



Protocol class	Network type	Name
0	A	Simple class
1	B	Basic error recovery class
2	A	Multiplexing class
3	B	Error recovery and multiplexing class
4	C	Error detection and recovery class

Fig. 6-8. Transport protocol classes.

- 
- Classe 0: é a classe mais simples, ela configura uma conexão para cada requisição de conexão de transporte e assume que esta não cometerá erros.
  - Classe 1: é igual a classe 0, exceto que foi desenvolvida para se recuperar de N-RESETs
  - Classe 2: como a classe 0, foi desenvolvida para trabalhar com conexões confiáveis, porém nela duas ou mais conexões de transporte podem ser enviadas na mesma conexão de rede.
  - Classe 3: é o equivalente das classes 1 e 2.
  - Classe 4: foi desenvolvida para o tipo de rede C, então é capaz de lidar com perdas, duplicações e adulterações de pacotes, N-RESETs, e qualquer outro tipo de erro.



## 6.1.5 - Elementos dos protocolos de transporte

- As características exatas providas por um protocolo de transporte depende no ambiente em que este opera, e o tipo de serviço que este provém.
- Temos uma lista com elementos básicos comuns em muitos protocolos de transporte. Mas esta não deve ser considerada literalmente, pois os detalhes dos protocolos podem variar de acordo com as classes.



Protocol element	Class				
	0	1	2	3	4
Connection establishment	x	x	x	x	x
Connection refusal	x	x	x	x	x
Assignment to network connection	x	x	x	x	x
Splitting long messages into TPDU's	x	x	x	x	x
Association of TPDU's with connection	x	x	x	x	x
TPDU transfer	x	x	x	x	x
Normal release	x	x	x	x	x
Treatment of protocol errors	x	x	x	x	x
Concatenation of TPDU's to the user		x	x	x	x
Error release	x		x		
TPDU numbering		x	o	x	x
Expedited data transfer		o	o	x	x
Transport layer flow control			o	x	x
Resynchronization after a RESET		x		x	x
Retention of TPDU's until ack		x		x	x
Reassignment after network disconnect		x		x	x
Frozen references		x		x	x
Multiplexing			x	x	x
Use of multiple network connections					x
Retransmission upon timeout					x
Resequencing of TPDU's					x
Inactivity timer					x
Transport layer checksum					o

x = present

o = optional

(blank) = absent



## 6.2 - Gerenciamento de Conexão

- Endereçamento
- Estabelecendo Conexão
- Terminando Conexão



## 6.2 - Gerenciamento de Conexão

### Siglas

- TPDU(Transport Protocol Data Unit)
- NSAP(Network Services Access Point)
- TSAP(Transport Services Access Point)

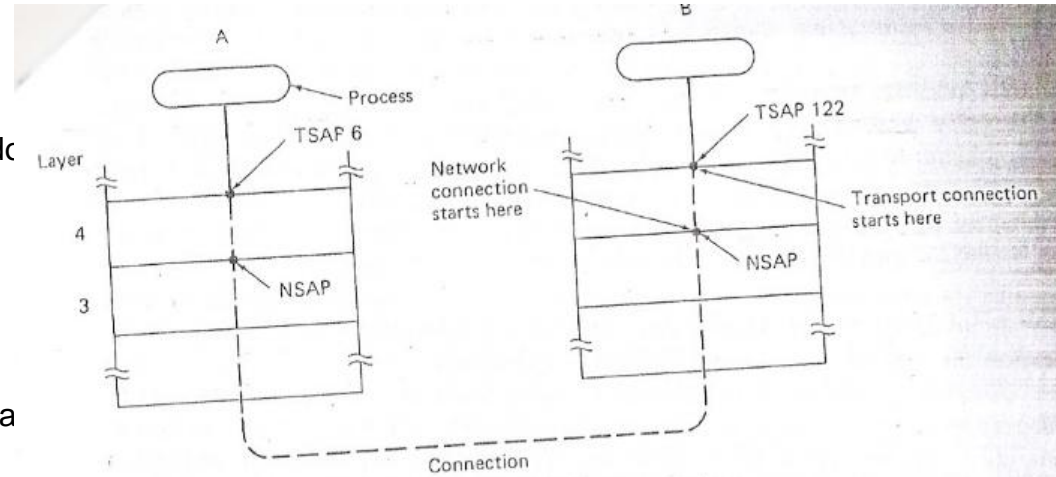


## 6.2.1 - Endereçamento

- Necessidade de especificar qual usuário remoto deve-se conectar.
- Método normalmente utilizado são os TSAPs os quais podem ser anexados aos processos e esperar conexão.

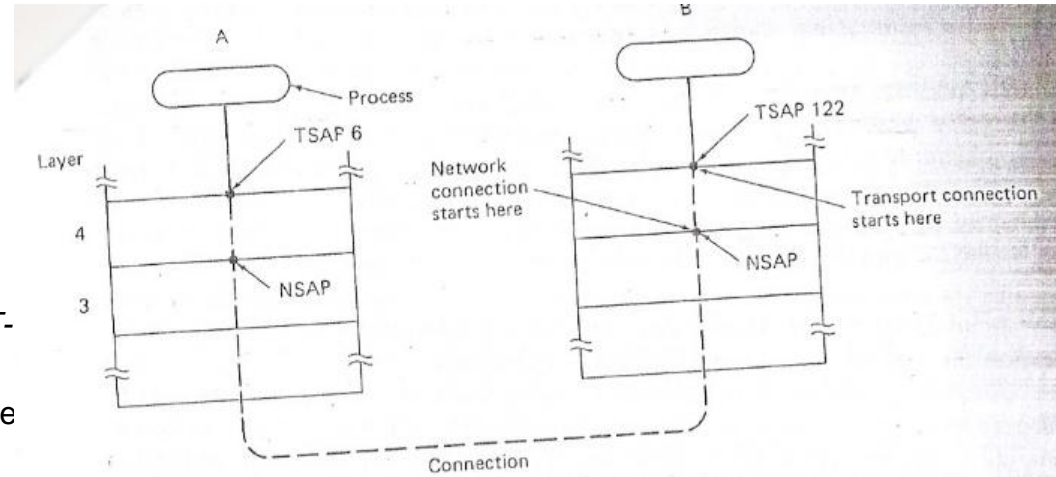
## 6.2.1 - Endereçamento

1. Um processo de servidor time-of-day, na máquina B anexa-se no TSAP 122 esperando um *T-CONNECT.indication*.
2. Um processo na máquina A quer descobrir quais horas são, então emite um *T-CONNECT.request* especificando TSAP 6 como origem e TSAP 122 como destino.
3. A entidade de transporte em A seleciona uma NSAP em sua máquina e no destinatário e configura uma conexão.



## 6.2.1 - Endereçamento

4. A primeira coisa que a entidade de transporte em A diz à B é: “Bom dia. Gostaria de estabelecer uma conexão de transporte entre TSAP 6 e TSAP 122. O que você me diz?”
5. A entidade de transporte em B então emite o *T-CONNECT.indication*, e se o servidor time-of-day em TSAP 122 for satisfeito, a conexão de transporte é estabelecida.



## 6.2.1 - Endereçamento

ARPANET protocolo de conexão inicial.

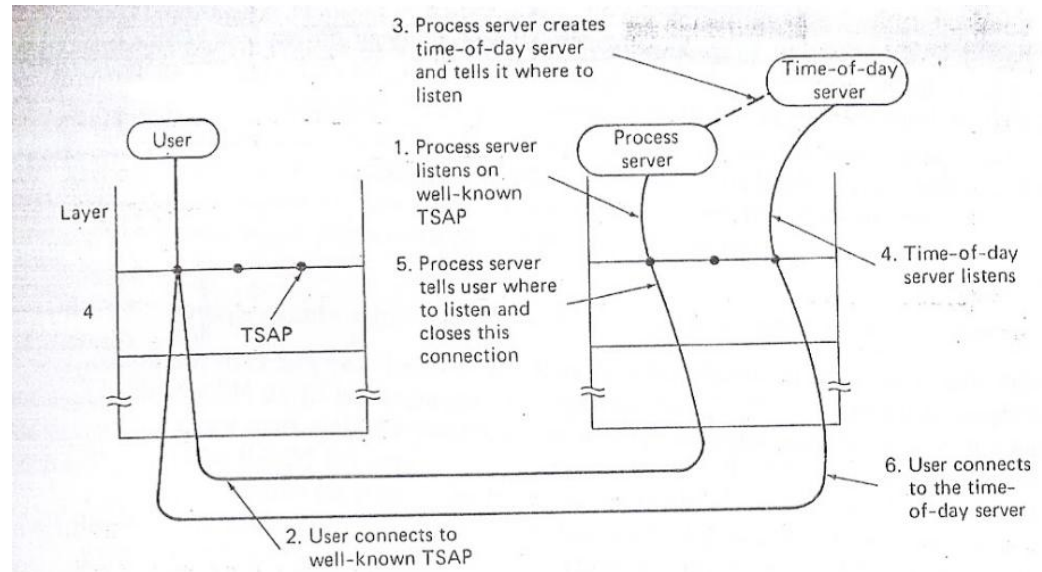


Fig. 6-11. How a user process in host A establishes a connection with a time-of-day server in host B.



## 6.2.1 - Endereçamento

- O protocolo inicial de conexão ARPANET funciona bem para servidores que são criados como necessário.
- Um servidor de arquivos por exemplo deve ter seu hardware especial e não pode ser criado em “durante a conexão”.
- Para isso, é criado o *name server* ou *directory server*. Para encontrar o endereço TSAP correspondente a um serviço, o usuário conecta-se com o *name server* enviando uma mensagem para o mesmo especificando o serviço, e o *name server* envia de volta o TSAP referente ao serviço especificado.





## 6.2.1 - Endereçamento

- Como uma entidade local de transporte saberá qual máquina está localizado algum determinado endereço TSAP?
- Qual NSAP usar para configurar uma conexão de rede para a entidade de transporte remota que gerencia o TSAP requisitado?
- A resposta está na estrutura utilizada para os endereços TSAP. Uma possível estrutura são os endereços hierárquicos.



## 6.2.1 - Endereçamento

Um endereço TSAP universal seria:

Address = <galaxy><star><planet><country><network><host><port>

Uma alternativa para o endereço hierárquico é o endereço de espaço plano. Mas, nesse caso é necessário um segundo nível de mapeamento para localizar a máquina necessária.



## 6.2.2 - Estabelecendo conexão

- Tem-se certa dificuldade em estabelecer uma conexão apesar de parecer fácil.
- O maior problema está na existência de pacotes duplicados e atrasados.
- A melhor solução está em destruir pacotes antigos presentes na sub rede. Se for possível ter certeza que nenhum pacote “vive” mais que um determinado tempo, o problema se torna gerenciável.



## 6.2.2 - Estabelecendo conexão

- O tempo de vida do pacote pode ser restrito em um máximo conhecido, usando uma das seguintes técnicas:
  1. Design de sub rede restrito.
  2. Contador de pulos em cada pacote.
  3. Estampa de tempo em cada pacote.



## 6.2.2 - Estabelecendo conexão

- Com a vida dos pacotes limitada é possível elaborar um ótimo método de estabelecer conexão, no entanto, o mesmo possui suas falhas.
- A ideia básica desse método é garantir que 2 TPDU's idênticos nunca serão únicos no mesmo tempo.



## 6.2.2 - Estabelecendo conexão

- Quando uma conexão é estabelecida, os  $k$  bits menos significativos do clock são utilizados como número de sequência inicial.
- O problema ocorre quando um *host* falha. Quando o mesmo retorna, sua entidade de transporte não sabe onde estava no número de sequência.

## 6.2.2 - Estabelecendo conexão

- Para prevenir o problema de replicação de TPDUs com mesmo número de sequência, previne-se os números de sequência serem usados em um determinado tempo  $T$  antes de ser usado um potencial número de sequência inicial.
- As combinações ilegais do tempo e seus números de sequência são mostrados nos gráficos como *forbidden region*.
- Seguindo essa lógica conclui-se que a taxa máxima de dados em qualquer conexão deve ser de um TPDU por tick de clock.
- O método baseado em clock resolve o problema de TPDUs atrasados e duplicados, desde que exista uma conexão estabelecida.

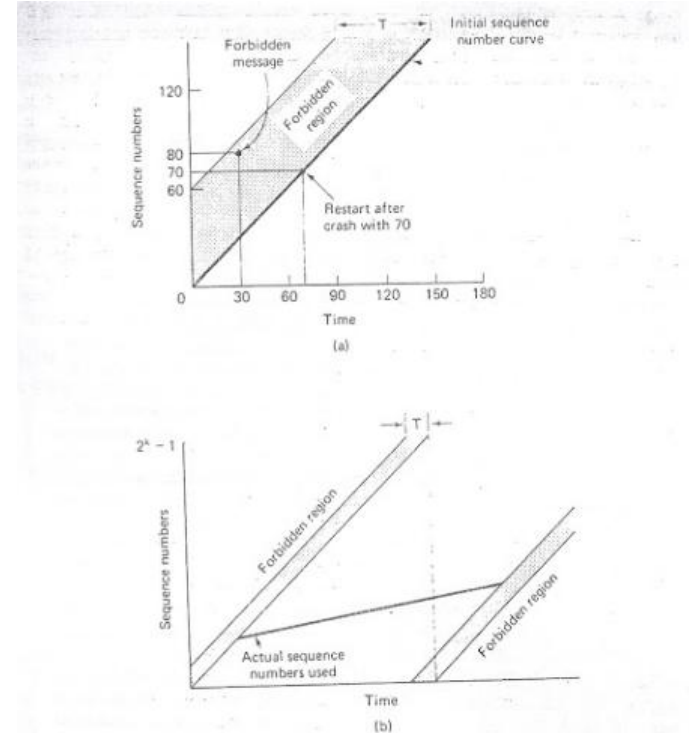


Fig. 6-12. (a) TPDUs may not enter the forbidden region. (b) The resynchronization problem.

## 6.2.3 - Estabelecendo conexão

- Two-army problem

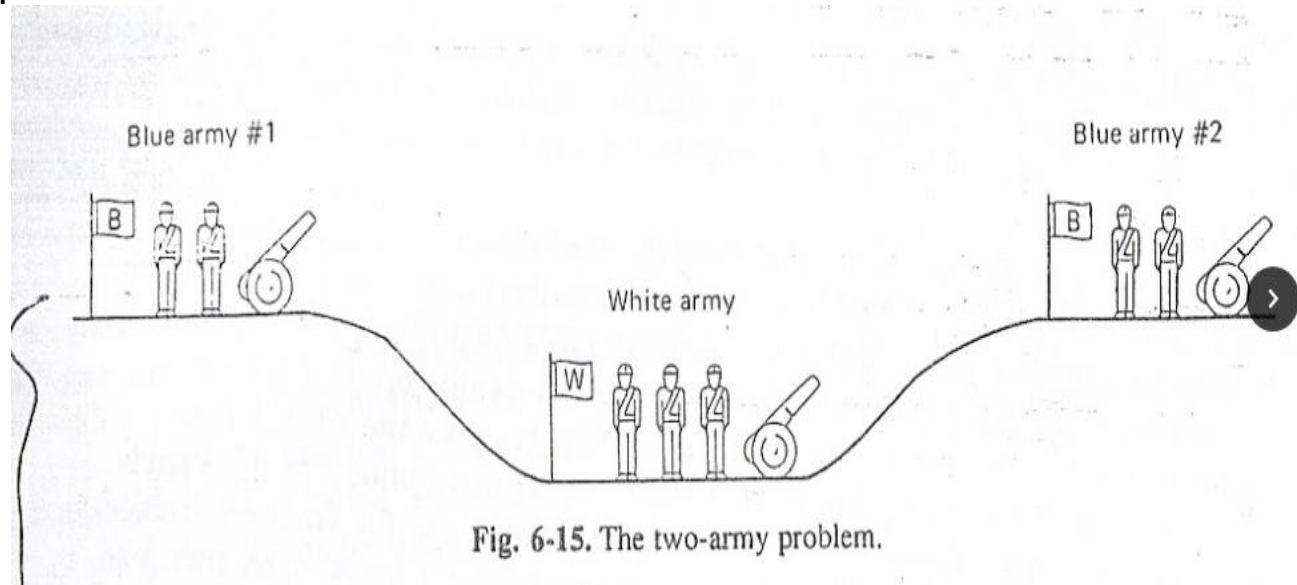


Fig. 6-15. The two-army problem.



## 6.2.2 - Estabelecendo conexão

- Three-way handshake, esse protocolo de estabelecimento de conexão não requer que os dois lados comecem com o mesmo número de sequência, então pode ser usado um método de sincronização em vez do método de clock global.

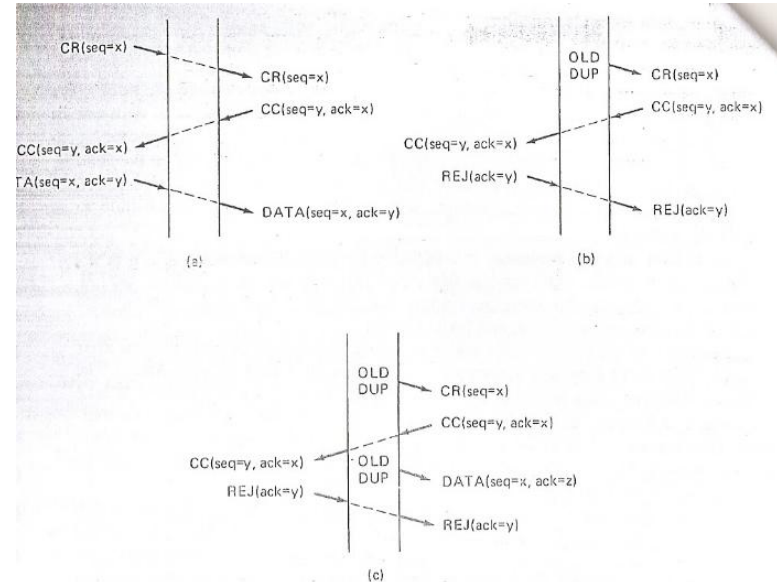


Fig. 6-13. Three protocol scenarios for establishing a connection using a three-way handshake. (a) Normal operation. (b) Old duplicate  $CR$  appearing out of nowhere. (c) Duplicate  $CR$  and duplicate ACK.



## 6.2.3 - Liberando conexão

- Liberar uma conexão é bem mais fácil que estabelecer uma.
- 3 formas básicas de liberar uma conexão
- Todas podem resultar em perda de dados.

## 6.2.3 - Liberando conexão

Three-way handshake protocol for releasing a connection.

- O único possível problema desse protocolo é falhar a DR inicial e todas as  $n$  retransmissões serem perdidas.
- Fazendo com que o emissor desista e desconecte enquanto o receptor continua na rede, criando uma conexão meio aberta.

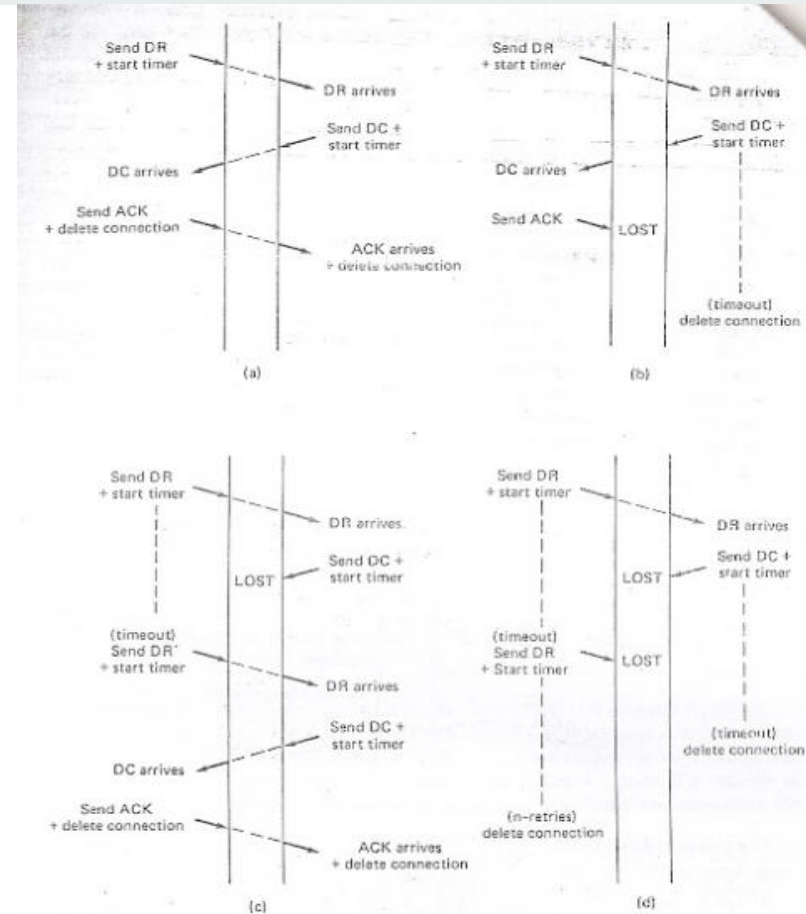


Fig. 6-16. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) DC lost. (d) DC lost and subsequent DRs lost.




### 6.2.3 - Liberando conexão


- Para resolver esse problema (tempo expirar), mandar um *dummy* TPDU para manter o outro lado conectado. De forma que, se muitos TPDU's forem perdidos um lado se desconectará seguido do outro.




## 6.2.4. Gerenciamento de Conexão Baseado em Timer


- Vamos dar um passo para trás. O problema original que apresentamos foi como evitar o pesadelo de pacotes antigos e/ou duplicados contendo CR, DATA e DR TPDUs que aparecia de repente do lado algum e sendo aceito como legítimo.


- 
- ... O que a Fletcher e a Watson fizeram, é providenciar para que a entidade de transporte se abstenha de excluir informações sobre uma conexão até que todas as TPDUs relacionadas a ela tenham desaparecido. Assim, em seu esquema, a tabela de uma entidade de transporte sobre uma conexão não é excluída quando a conexão é liberada, mas quando um intervalo de tempo cuidadosamente escolhido expirou.


- 
- O coração de seu esquema é este: quando um remetente deseja enviar um fluxo de TPDU's consecutivas para um receptor, ele cria um registro de conexão internamente. Este registro de conexão mantém o controle de quais TPDU's foram enviados, que reconhecimentos foram recebidos, e assim por diante. Sempre que um registro de conexão é criado, um temporizador é iniciado. Sempre que uma TPDU é enviada usando uma gravação de conexão criada anteriormente, o temporizador é iniciado novamente. Se o temporizador expirar (o que significa que nada foi enviado para um determinado intervalo), o registro de conexão é excluído.


- 
- A primeira TPDU no fluxo contém uma bandeira de 1 bit chamada DRF (Data Run Flag), que a identifica como a primeira em uma série de TPDUs. Quando qualquer TPDU é enviada, um temporizador é iniciado. Se a TPDU for reconhecida, o temporizador é interrompido. Se o temporizador for desligado, a TPDU é retransmitida. Se, após n retransmissões, ainda não houver confirmação, o remetente abandona




- 
- Quando uma TPDU com o conjunto de sinalização DRF chega ao receptor, o receptor assinala seu número de sequência e cria um registro de conexão. As TPDU's subsequentes só serão aceitas se estiverem em sequência.
  - Em outras palavras, um registro de conexão só é criado quando um TPDU com um conjunto de DRF chega. Sempre que uma TPDU chega em sequência, ela pode ser passada para o usuário do transporte e uma confirmação retornada ao remetente.

- 
- Vamos primeiro considerar um caso simples de como este protocolo funciona. Uma sequência de TPDU's é enviada e todas são recebidas por ordem e reconhecidas. Quando o remetente obtém a confirmação do TDPU final enviado e vê o sinalizador ARF, ele para todos os temporizadores de retransmissão. Se não houver mais dados do usuário do transporte, eventualmente o registro de conexão do receptor expira e depois o remetente também.
  - Agora, considere o que acontece se a TPDU com a bandeira DRF for perdida. O receptor não criará um registro de conexão. Eventualmente, o remetente expirará e retransmitirá a TPDU, repetidamente, se necessário, até que seja reconhecido.

- 
- Suponha que um fluxo de TPDU's seja enviado e recebido corretamente, mas alguns das mensagens de confirmação são perdidas. As retransmissões subsequentes também são perdidas, de modo que o registro de conexão do receptor expira e é excluído. O que fazer para evitar que uma duplicata antiga da TPDU com a flag DRF apareça agora no receptor e desencadeie um novo registro de conexão? ???


- 
- Agora, vejamos o que acontece se o usuário do transporte enviar mensagens em rajadas. Suponha que uma explosão de TPDUs seja enviada e todos sejam reconhecidos. Quando o usuário do transporte finalmente chega ao envio de uma nova mensagem, uma das três situações deve aguentar:
    1. Tanto o remetente quanto o receptor ainda possuem seus registros de conexão.
    2. O remetente tem seu registro de conexão, mas o receptor não.
    3. Ambos os registros de conexão foram excluídos.

- 
- Em resumo, este protocolo possui uma mistura interessante de propriedades sem conexões e orientadas para a conexão. A troca mínima é duas TPDUs, o que torna tão eficiente como um protocolo sem conexão para aplicativos de resposta de consulta. Ao contrário de um protocolo sem conexão, no entanto, se se verificar que uma sequência de TPDUs deve ser enviada, elas devem ser entregues em ordem. E também temos que, as conexões são liberadas automaticamente pelo uso inteligente, dos temporizadores.



## 6.2.5. Controle de Fluxo e Buffering

- Tendo examinado o estabelecimento de conexão e lançado em alguns detalhes, vejamos agora como as conexões são gerenciadas enquanto estão em uso. Uma das principais questões surgiu antes: controle de fluxo
- A semelhança básica é que em ambas as camadas é necessária uma janela deslizante ou outro esquema em cada conexão para evitar que um transmissor rápido seja superado por um receptor lento.

- 
- A principal diferença é que o IMP geralmente tem relativamente poucas linhas enquanto o host pode ter várias conexões. Essa diferença torna impossível implementar a estratégia de buffer de ligação de dados na camada de transporte.

- Na camada de link de dados, o lado de envio deve armazenar os quadros de saída porque eles podem ter que ser retransmitidos. Se a sub-rede fornecer serviço de datagrama, a entidade de transporte de envio também deve armazenar um buffer e pelo mesmo motivo.

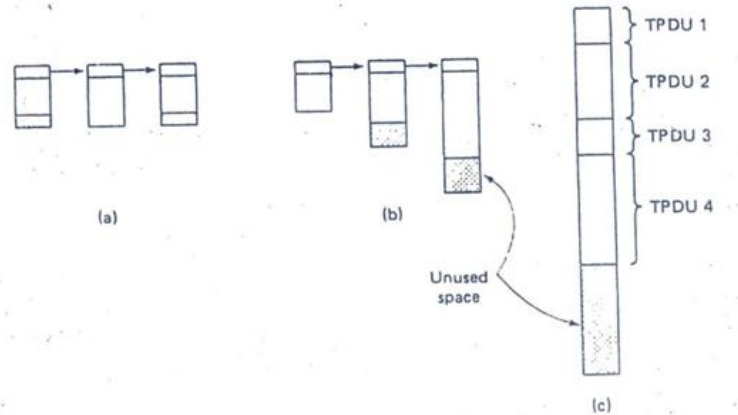




Fig. 6-17. (a) Chained fixed-size buffers. (b) Chained variable-size buffers. (c) One large circular buffer per connection.



- 
- Em resumo, se o serviço de rede não é confiável (ou seja, digite B ou C), o remetente deve armazenar todas as TPDU's enviadas, assim como na camada de link de dados. No entanto, com serviço de rede confiável (tipo A), outros trade-offs tornam-se possíveis. "em particular, se o Remetente sabe que o receptor sempre possui espaço de buffer, não precisa reter cópias das TPDU's que envia. No entanto, se o receptor não puder garantir que todas as TPDU's entrantes serão aceitas, o remetente terá que amortecer de qualquer maneira. No último caso, o remetente não pode confiar na confirmação da camada de rede, porque o reconhecimento significa apenas que a TPDU chegou, não que tenha sido aceita.




- Mesmo que o receptor tenha concordado em fazer o buffer, ainda resta a questão do tamanho do buffer
- Se o tamanho do buffer for escolhido menos do que o tamanho máximo da TPDU, serão necessários vários buffers para TPDU's longas, com a complexidade do atendimento.
- ...O trade-off ideal entre buffer de origem e buffer de destino depende do tipo de tráfego transportado pela conexão


- 
- À medida que as conexões são abertas e fechadas, e à medida que o padrão de tráfego muda, o remetente e o receptor precisam ajustar dinamicamente suas alocações de buffer.
  - Consequentemente, o protocolo de transporte deve permitir que um host de envio solicite espaço de buffer na outra extremidade. Os buffers podem ser alocados por conexão, ou coletivamente, para todas as conexões que funcionam entre os dois hosts.
  - ... Uma maneira razoavelmente geral de gerenciar a alocação de buffer dinâmico é desacoplar o buffer de confirmação dos reconhecimentos, em contraste com os protocolos de janela deslizante do Capítulo 4.


- A Figura 6-18 mostra um exemplo de como o gerenciamento de janela dinâmico pode funcionar em uma sub-rede de datagramas com números de sequência de 4 bits.


	A	Message	B	Comments
1	→	< request 8 buffers >	→	A wants 8 buffers
2	←	< ack = 15, buf = 4 >	←	B grants messages 0-3 only
3	→	< seq = 0, data = m0 >	→	A has 3 buffers left now
4	→	< seq = 1, data = m1 >	→	A has 2 buffers left now
5	→	< seq = 2, data = m2 >	...	Message lost but A thinks it has 1 left
6	←	< ack = 1, buf = 3 >	←	B acknowledges 0 and 1, permits 2-4
7	→	< seq = 3, data = m3 >	→	A has 1 buffer left
8	→	< seq = 4, data = m4 >	→	A has 0 buffers left, and must stop
9	→	< seq = 2, data = m2 >	→	A times out and retransmits
10	←	< ack = 4, buf = 0 >	←	Everything acknowledged, but A still blocked
11	←	< ack = 4, buf = 1 >	←	A may now send 5
12	←	< ack = 4, buf = 2 >	←	B found a new buffer somewhere
13	→	< seq = 5, data = m5 >	→	A has 1 buffer left
14	→	< seq = 6, data = m6 >	→	A is now blocked again
15	←	< ack = 6, buf = 0 >	←	A is still blocked
16	...	< ack = 6, buf = 4 >	←	Potential deadlock

Fig. 6-18. Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU.

- 
- Podem surgir problemas potenciais com esquemas de alocação de buffer desse tipo em redes de datagramas se as TPDUs de controle puderem se perder. Olhe para a linha 16. B alocou mais buffers para A, mas a TPDU de alocação foi perdida. Uma vez que as TPDUs de controle não são sequenciadas nem expiraram, A está agora bloqueada. Para evitar esta situação, cada host deve enviar periodicamente TPDUs de controle, dando o status de confirmação e buffer em cada conexão. Dessa forma, o impasse estará quebrado, mais cedo ou mais tarde

- 
- Quando o espaço do buffer não limita mais o fluxo máximo, outro gargalo irá aparecer: a capacidade de carga da sub-rede. Se o IMPs adjacente pode trocar na maioria dos  $x$  frames / seg e há  $k$  caminhos disjuntos entre um par de hosts, não há nenhuma maneira de que esses hosts possam trocar mais do que  $kx$  TPDUs / seg, independentemente de quanto tempo de buffer esteja disponível em cada extremidade.

- 
- O que é necessário é um mecanismo baseado na capacidade de carga da sub-rede e não na capacidade de armazenamento do receptor.
  - Claramente, o mecanismo de controle de fluxo deve ser aplicado no remetente, para evitar que ele tenha muitas TPDUs não reconhecidas pendentes ao mesmo tempo

- 
- Belsnes (1975) propôs o uso de um esquema de controle de fluxo de janela deslizante no qual o remetente ajusta dinamicamente o tamanho da janela
  - A capacidade de carga pode ser determinada simplesmente contando o número de TPDU reconhecido durante algum período de tempo e depois dividindo pelo período de tempo. Durante a medição, o remetente deve enviar o mais rápido possível, para garantir que a capacidade de carga da rede, e não a baixa taxa de entrada, seja o fator que limita a taxa de confirmação. para coincidir com a capacidade de carga da rede.





## 6.2.6. Multiplexação

- Multiplexar várias conversas em conexões, circuitos virtuais e links físicos desempenha um papel em várias camadas da arquitetura de rede. Na camada de transporte, a necessidade de multiplexação pode surgir de várias maneiras. Por exemplo, em redes que usam circuitos virtuais dentro da sub-rede, cada conexão aberta consome algum espaço de tabela nas IMPs durante toda a duração da conexão.

uma estrutura de preços que penaliza fortemente as instalações para ter muitos circuitos virtuais abertos por longos períodos de tempo é tornar a multiplexação de diferentes conexões de transporte para a mesma conexão de rede atrativa. Esta forma de multiplexação, chamada de multiplexação ascendente, é mostrada na Fig. 6-19 (a)

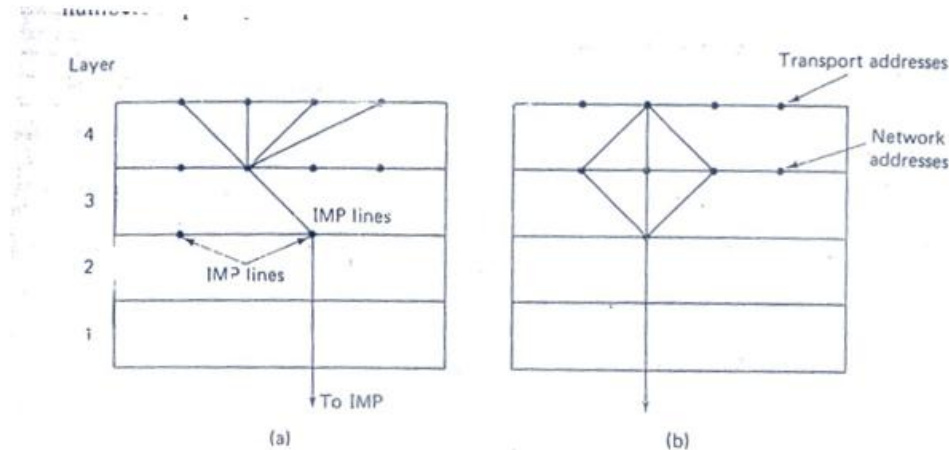




Fig. 6-19. (a) Upward multiplexing. (b) Downward multiplexing.



Quando o tempo de conexão forma o principal componente da conta do transportador, cabe à camada de transporte agrupar as conexões de transporte de acordo com seu destino e mapear cada grupo no número mínimo de conexões de rede. Se existem muitas conexões de transporte mapeado em uma conexão de rede, o desempenho será fraco, porque a janela geralmente estará cheia, e os usuários terão de esperar sua vez de enviar uma mensagem. Se muitas poucas conexões de transporte estiverem mapeadas em uma conexão de rede, o serviço será caro



A multiplexação também pode ser útil na camada de transporte por outro motivo, relacionado às decisões técnicas da operadora ao invés de decisões de preços de operadoras. Suponha, por exemplo, que um determinado usuário importante precise de uma conexão de banda larga de tempos em tempos. Se a sub-rede impõe um controle de fluxo de janela deslizante com um número de sequência de 3 bits, o usuário deve parar de enviar o quanto antes como sete pacotes são pendentes, e deve aguardar os pacotes se propagar para o host remoto e ser reconhecido.




## 6.2.7. Crash Recovery


Se hosts e IMPs estão sujeitos a falhas, a recuperação desses acidentes se torna um problema. Se a camada de rede emitir um N-RESET, por exemplo, as entidades de transporte. deve trocar informações após o acidente para determinar quais TPDUs foram recebidas e quais não foram. Na verdade, depois que um host A do choque pode pedir ao host B: "Eu tenho quatro TPDUs não reconhecidas pendentes, 2, 3, 4 e 5, você recebeu alguma delas?" Com base na resposta, A pode retransmitir as TPDUs apropriadas, desde que tenha mantido cópias delas.



Um problema mais incômodo é como se recuperar de falhas do host. Para ilustrar a dificuldade, vamos assumir que um host, o remetente, está enviando um arquivo longo para outro host, o receptor, usando um simples protocolo de parada e espera. A camada de transporte no host receptor simplesmente passa as TPDU's recebidas para o usuário do transporte, uma a uma. Em parte através da transmissão, o receptor falha. Quando ela volta, suas tabelas são reinicializadas, então não sabe mais exatamente onde estava. Na tentativa de recuperar seu status anterior, o receptor pode enviar uma TPDU de transmissão para todos os outros hosts, anunciando que acabou de bater e solicitar aos outros hosts que informassem sobre o status de todas as conexões abertas. O remetente pode estar em um dos dois estados: uma TPDU pendente, S1, ou nenhuma TPDU pendente, S0.



À primeira vista, parece óbvio que o remetente deve retransmitir apenas se tiver uma TPDU não reconhecida em destaque (ou seja, está no estado SI) quando descobre o acidente. No entanto, uma inspeção mais próxima revela dificuldades com essa abordagem ingênua.




Considere, por exemplo, a situação em que o host recebedor primeiro envia uma confirmação, e então, quando a confirmação foi enviada, executa a gravação. Escrever uma TPDU no fluxo de saída e enviar uma confirmação são considerados dois eventos indivisíveis distintos que não podem ser realizados simultaneamente. Se ocorrer uma falha após o recebimento da confirmação, mas antes que a escrita tenha sido feita, o outro host receberá a confirmação e, portanto, estará no estado S0 quando o anúncio da recuperação de falhas chegar. O remetente, portanto, não retransmite, pensando que a TPDU chegou corretamente, levando a uma TPDU em falta.







- Imagine que a gravação foi feita, mas a falha ocorre antes que a confirmação possa ser enviada. O remetente estará no estado S1 e assim retransmitirá, levando a uma TPDU duplicada não detectada no fluxo de saída. Não importa como o remetente e o receptor estão programados, sempre há situações em que o protocolo não recupera corretamente. O receptor pode ser programado de duas maneiras: reconhecer primeiro ou escrever primeiro. O remetente pode ser programado de uma das quatro maneiras: sempre retransmite a última TPDU, nunca retransmite
- A última TPDU, retransmitir apenas no estado S0, ou retransmitir apenas no estado S1. Isso dá oito combinações, mas, como veremos, para cada combinação há algum conjunto de eventos que faz com que o protocolo falhe.



Os eventos de árvore são possíveis no receptor: enviar uma confirmação (A), escrever para o processo de saída (W) e falhar (C). Os três eventos podem ocorrer em seis ordens diferentes: AC (W), AWC, C (AW), C (WA), WAC e WC (A), onde os parênteses são usados para indicar que nem A nem W podem seguir C (ou seja, uma vez que ele caiu, caiu). A Figura 6-20 mostra todas as oito combinações de estratégia de remetente e receptor e as sequências de eventos válidas para cada uma delas. Observe que, para cada estratégia, há uma sequência de eventos que faz com que o protocolo se comporte incorretamente. Por exemplo, se o remetente sempre retransmitir, o evento AWC gerará uma duplicata não detectada, mesmo que os outros dois eventos funcionem corretamente.



A conclusão é inescapável: sob nossas regras básicas de eventos não simultâneos, o host crash / recovery não pode ser transparente para camadas mais altas. Colocando em termos mais gerais, esse resultado pode ser atualizado, pois a recuperação de uma camada N de bloqueio só pode ser feita pela camada  $N + 1$  e, em seguida, somente se a camada mais alta conservar informações de status suficientes. Conforme mencionado acima, a camada de transporte pode se recuperar de falhas na camada de rede, desde que cada extremidade de uma conexão acompanhe o local onde está.



Este problema nos coloca na questão do que realmente significa um chamado reconhecimento de ponta a ponta. Em princípio, o protocolo de transporte é de ponta a ponta e não está encadeado como as camadas inferiores.



## 6.3. Um Simples Protocolo Sob o X.25

- Esse capítulo tem como objetivo tornar as idéias discutidas até o momento mais concretas. Estaremos estudando a camada de transporte em detalhes.
- O exemplo escolhido é realístico, mas ainda simples o suficiente para ser facilmente entendido.
- Os serviços abstratos são as primitivas OSI orientadas a conexão que só adiciona complexidade, sem prover nenhuma nova visão sobre como a camada de transporte funciona.



## 6.3.1. Exemplo do Serviço de Primitivas

- Nosso primeiro problema é como expressar o transporte de primitivas OSI em pascal.
- Para fazer o `CONNECT.request` é fácil, só precisamos ter uma biblioteca `connect` que possa ser chamada com parâmetros necessários apropriados para estabelecer uma conexão.
- Mas para fazer um `connect.indication` é mais difícil, como sinalizamos o usuário de transporte que há uma chamada sendo recebida?



### 6.3.1. Exemplo do Serviço de Primitivas

- A primitiva `CONNECT.indication` é um ótimo meio de mostrar como funciona para telefones, mas não é um bom meio para mostrar como funcionam em computadores.
- No nosso modelo há dois processos disponíveis, `listen` e `connect`.  
Quando um processo quer estar pronto para aceitar chamadas ele chama o `listen` especificando um TSAP em particular para atender.
- O processo então é bloqueado até que algum processo remoto estabeleça conexão ao seu TSAP.



### 6.3.1. Exemplo do Serviço de Primitivas

- Outro procedimento que é o connect, pode ser usado quando um processo quer iniciar uma conexão.
- A fonte especifica a TSAP local e a remota, e são bloqueadas quando a camada de transporte tenta estabelecer uma conexão.
- Quando a conexão é bem sucedida, ambos são desbloqueados e podem começar a trocar dados.





### 6.3.1. Exemplo do Serviço de Primitivas

- Esse modelo é muito assimétrico, sendo que quando um está executando um listen esperando algo acontecer, o outro lado está ativo.
- Mas o que fazer se o lado ativo começar primeiro?
- Uma estratégia é fazer a tentativa de conexão falhar caso não tenha ninguém para atender à conexão na TSAP remota.
- Outra estratégia é ter um bloqueio no initiator até que algum listener para ser conectado apareça.



### 6.3.1. Exemplo do Serviço de Primitivas

- Uma forma também é segurar o pedido de conexão no local recebido por um intervalo de tempo.
- Se um processo naquele host chamar um listen antes do temporizador acabar o tempo, a conexão será estabelecida, caso contrário é rejeitado e o caller é desbloqueado.
- Para desfazer uma conexão usaremos o disconnect. Quando os dois lados forem desconectados a conexão será encerrada.



### 6.3.1. Exemplo do Serviço de Primitivas

- A transmissão de dados tem o mesmo problema de iniciar uma conexão apesar que T-DATA.request pode ser implementado diretamente pela chamada de uma biblioteca de processos, T-DATA.indication não pode ser.
- Nós usaremos a mesma solução que usamos para estabelecer uma conexão para solucionar o problema de transmissão de dados, uma chamada passiva recebe que bloqueia até que a mensagem chegue.



### 6.3.1. Exemplo do Serviço de Primitivas

- São 5 primitivas que definem esse serviço: CONNECT, LISTEN, DISCONNECT, SEND e RECEIVE. Cada primitiva corresponde a uma biblioteca de processos que executa a primitiva. Os parâmetros para cada primitiva são:

connum = CONNECT(local, remote)

connum = LISTEN(local)

status = DISCONNECT(connum)

status = SEND(connum, buffer, bytes)

status = RECEIVE(connum, buffer, bytes)



### 6.3.1. Exemplo do Serviço de Primitivas

- A primitiva CONNECT tem a TSAP local e uma TSAP remota e faz a conexão entre as duas. Se é bem sucedido retornará a connum um número não negativo. Se falhar retornará um número negativo. Alguns motivos para falha são: destino atualmente sendo utilizado, host remoto caiu, endereço local ilegal, endereço remoto ilegal.
- A primitiva LISTEN diz para o caller aceitar a conexão na TSAP indicada. O usuário da primitiva é bloqueado até que haja uma tentativa de conexão. Não há timeout.



### 6.3.1. Exemplo do Serviço de Primitivas

- A primitiva DISCONNECT termina um transporte de conexão, o parâmetro connum indica qual. Possíveis erros: connum pertence a outro processo, ou connum não é um identificador de conexão válido. Retornará 0 se for bem sucedido, ou um código de erro se houver falhas.
- A primitiva SEND transmite o conteúdo do buffer como uma mensagem para a conexão de transporte indicada. Possíveis erros retornados para status, são: sem conexão, endereço de buffer ilegal, ou contagem negativa.



### 6.3.1. Exemplo do Serviço de Primitivas

- A primitiva RECEIVE indica o desejo do caller de aceitar os dados. O tamanho da mensagem a ser recebida é em bytes. Se o acesso remoto tiver a conexão encerrada ou o endereço do buffer for ilegal, o status é definido com um código de erro.



## 6.3.2. Exemplo de Entidade de Transporte

- A camada de transporte faz uso das primitivas dos serviços de rede para mandar e receber TPDU's. Assim como as primitivas de serviços de rede não podem ser mapeadas diretamente pela bibliotecas de processos, as primitivas de serviço de transporte também não podem. Nesse exemplo, evitamos esse problema através do X.25 como interface da camada de rede.
- Cada TPDU será carregado em um pacote, e cada pacote corresponderá a um TPDU.





## 6.3.2. Exemplo de Entidade de Transporte

- No exemplo a seguir nós vamos assumir que o X.25 é confiável, sem perder pacotes ou resetar o circuito. Veremos como é a implementação do nosso serviço de transporte.
- Esse determinado programa ( que é o código de uma entidade de transporte ) pode ser parte do sistema operacional do host ou pode ser pacotes de rotina de biblioteca sendo executado no endereço de espaço do usuário.
- O exemplo a seguir foi programado como um pacote de biblioteca, mas as mudanças para fazerem parte de um sistema operacional são mínimas. ( Primariamente como buffer são acessados )

```

const MaxConn = ...; MaxMsg = ...; MaxPkt = ...;
TimeOut = ...; cred = ...;
q0 = 0; q1 = 1; m0 = 0; m1 = 1; ok = 0;
ErrFull = -1; ErrReject = -2; ErrClosed = -3; LowErr = -3;

type bit = 0..1;
TransportAddress = integer;
ConnId = 0 .. MaxConn;      {connection identifier}
PktType = (CallReq, CallAcc, ClearReq, ClearConf, DataPkt, credit);
cstate = (idle, waiting, queued, established, sending, receiving, disconnecting)
message = array[0 .. MaxMsg] of 0 .. 255;
msgptr = 1message;          {pointer to a message}
ErrorCode = LowErr .. 0;
ConnIdOrErr = LowErr .. MaxConn;
PktLength = 0 .. MaxPkt;
packet = array[PktLength] of 0 .. 255;

var ListenAddress: TransportAddress; {local address being listened to}
ListenConn: ConnId;                {connection identifier for listen}
data: packet;                       {scratch area for packet data}
conn: array[ConnId] of record
  LocalAddress, RemoteAddress: TransportAddress;
  state: cstate;                    {state of this connection}
  UserBufferAddress: msgptr;        {pointer to receive buffer}
  ByteCount: 0 .. MaxMsg;           {send/receive count}
  CtrReqReceived: boolean;          {set when CLEAR REQUEST packet received}
  timer: integer;                   {used to time out CALL REQUEST packets}
  credits: integer;                 {number of messages that may be sent}
end;

function listen(t: TransportAddress): ConnIdOrErr;
{User wants to listen for a connection. See if CALLREQ has already arrived.}
var i: integer; found: boolean;
begin
  i := 1;
  found := false;
  while (i <= MaxConn) and not found do
    if (conn[i].state = queued) and (conn[i].LocalAddress = t)
      then found := true
      else i := i + 1;
  if not found then
    begin {no CALLREQ is waiting. Go to sleep until arrival or timeout.}
      ListenAddress := t; sleep; i := ListenConn
    end;
  conn[i].state := established;      {connection is established}
  conn[i].timer := 0;                {timer is not used}
  listen := i;                       {return connection identifier}
  ListenConn := 0;                   {0 is assumed to be an invalid address}
  ToNet(i, q0, m0, CallAcc, data, 0) {tell net to accept connection}
end; {listen}

```

```

function connect(l, r: TransportAddress): ConnIdOrErr;
{User wants to connect to a remote process. Send CALLREQ packet.}
var i: Integer;
begin i := MaxConn;      {search table backwards}
  data[0] := r; data[1] := l; {CALL REQUEST packet needs these}
  while (conn[i].state <> idle) and (i > 1) do i := i - 1;
  if conn[i].state = idle then
    with conn[i] do
      begin
        {make a table entry that CALLREQ has been sent}
        LocalAddress := l; RemoteAddress := r; state := waiting;
        ClrReqReceived := false; credits := 0; timer := 0;
        ToNet(i, q0, m0, CallReq, data, 2);
        sleep; {wait for CALLACC or CLEARREQ}
        if state = established then connect := i;
        if ClrReqReceived then
          begin
            {other side refused call}
            connect := ErrReject;
            state := idle; {back to idle state}
            ToNet(i, q0, m0, ClearConf, data, 0)
          end
        end
      end
    else connect := ErrFull {reject CONNECT: no table space}
  end; {connect}

function send(cid: ConnId; bufptr: msgptr; bytes: integer): ErrorCode;
{User wants to send a message.}
var i, count: integer; m: bit;
begin
  with conn[cid] do
    begin
      {enter sending state}
      state := sending;
      ByteCount := 0;
      if (not ClrReqReceived) and (credits = 0) then sleep;
      if not ClrReqReceived then
        begin {credit available; split message into packets}
          repeat
            if bytes - ByteCount > MaxPkt
              then begin count := MaxPkt; m := 1 end
              else begin count := bytes - ByteCount; m := 0 end;
            for i := 0 to count - 1 do data[i] := bufptr↑[ByteCount + i];
            ToNet(cid, q0, m, DataPkt, data, count);
            ByteCount := ByteCount + count;
          until ByteCount = bytes; {loop until whole message sent}
          credits := credits - 1; {one credit used up}
          send := ok
        end
      else send := ErrClosed; {SEND failed: peer wants to disconnect}
      state := established
    end
  end; {send}
end;

```

```

function receive(cid: ConnId; bufptr: msgptr; var bytes: integer): ErrorCode;
{User is prepared to receive a message.}
begin
  with conn[cid] do
    begin
      if not ClrReqReceived then
        begin
          {connection still established; try to receive}
          state := receiving;
          UserBufferAddress := bufptr; ByteCount := 0;
          data[0] := cred; data[1] := 1;
          ToNet(cid, q1, m0, credit, data, 2); {send credit}
          sleep; {block awaiting data}
          bytes := ByteCount
        end;
        if ClrReqReceived then receive := ErrClosed else receive := ok;
        state := established
      end
    end; {receive}
function disconnect(cid: ConnId): ErrorCode;
{User wants to release a connection.}
begin
  with conn[cid] do
    if ClrReqReceived
      then begin state := idle; ToNet(cid, q0, m0, ClearConf, data, 0) end
      else begin state := disconnecting; ToNet(cid, q0, m0, ClearReq, data, 0) end;
    disconnect := ok
  end; {disconnect}
procedure PacketArrival;
{A packet has arrived, get and process it.}
var cid: ConnId; {connection on which packet arrived}
    q, m: bit;
    pctype: PkType; {CallReq, CallAcc, ClearReq, ClearConf, DataPkt, credit}
    data: packet; {data portion of the incoming packet}
    count: PkLength; {number of data bytes in packet}
    i: integer; {scratch variable}
begin
  FromNet(cid, q, m, pctype, data, count); {go get it}
  with conn[cid] do
    case pctype of
      CallReq: {remote user wants to establish connection}
        begin
          LocalAddress := data[0]; RemoteAddress := data[1];
          if LocalAddress = ListenAddress
            then begin ListenConn := cid; state := established; wakeup end
            else begin state := queued; timer := TimeOut end;
          ClrReqReceived := false; credits := 0
        end;
    end;

```

```

CallAcc:                                {remote user has accepted our CALL REQUEST}
begin
    state := established;
    wakeup
end;

ClearReq:                                {remote user wants to disconnect or reject call}
begin
    ClrReqReceived := true;
    if state = disconnecting then state := idle; {clear collision}
    if state in {waiting, receiving, sending} then wakeup
end;

ClearConf:                               {remote user agrees to disconnect}
state := idle;

credit:                                  {remote user is waiting for data}
begin
    credits := credits + data[1];
    if state = sending then wakeup
end;

Datapkt:                                 {remote user has sent data}
begin
    for i := 0 to count - 1 do UserBufferAddress↑[ByteCount + i] := data[i];
    ByteCount := ByteCount + count;
    if m = 0 then wakeup
end
end;
end; {PacketArrival}

procedure clock;
{The clock has ticked, check for timeouts of queued connect requests.}
var i: ConnId;
begin
    for i := 1 to MaxConn do
        with conn[i] do
            if timer > 0 then
                begin {timer was running}
                    timer := timer - 1;
                    if timer = 0 then
                        begin {timer has expired}
                            state := idle;
                            ToNet(i, q0, m0, ClearReq, data, 0)
                        end
                    end
                end
            end
        end
    end
end; {clock}

```



## 6.3.2. Exemplo de Entidade de Transporte

- Nesse exemplo, a "entidade de transporte" não é uma entidade separada mas parte do processo usuário. Se o usuário utilizar uma primitiva que causa bloqueamento, como um LISTEN, toda a entidade de transporte é bloqueada.
- Enquanto nesse design serve para um host que tem apenas um usuário se conectando, se houvesse mais usuários se conectando, seria mais natural ter uma entidade de transporte separada.



## 6.3.2. Exemplo de Entidade de Transporte

- A interface da camada de redes é por meio dos processos ToNet e o FromNet. Cada um tem 6 parâmetros, o identificador de conexão que mapeia um por um na rede de circuito virtual; o X.25 Q e M bits, que indica uma mensagem de controle e mais dados dessa mensagem são passados nos próximos pacotes respectivos; o tipo de pacote, escolhido pelo conjunto CALL REQUEST, CALL ACCEPTED, CLEAR REQUEST, CLEAR CONFIRMATION, DATA, e CREDIT; um ponteiro para os próprios dados, e o número de bytes dos dados.



## 6.3.2. Exemplo de Entidade de Transporte

- Em chamadas ToNet a entidade de transporte completa todos os parâmetros para que a camada de redes a leia. No FromNet, a camada de redes desmembra cada pacote para a entidade de transporte.
- Ao passar informações por parâmetro, estamos sendo poupados de entrar nos detalhes da camada de redes.
- Se a entidade de transporte tenta mandar um pacote para um circuito virtual de janela deslizando cheio, ele é suspenso em ToNet até que tenha espaço para ele.





## 6.3.2. Exemplo de Entidade de Transporte

- Em adição a esses mecanismos de suspensão, há também processos como o sleep e o wakeup chamada pela entidade de transporte. O sleep é chamado quando a entidade de transporte tem que esperar um evento externo ocorrer e fica bloqueado até que isso aconteça. Geralmente espera um pacote chegar. Depois disso, a entidade de transporte e o processo usuário para de executar.
- Cada estado sustentado pela entidade de transporte é sempre um desses 7 a seguir:



## 6.3.2. Exemplo de Entidade de Transporte

Idle - conexão ainda não estabelecida

Waiting - Conexão foi executada e CALL REQUEST enviado

Queued - CALL REQUEST chegou; listen não foi feito.

Established - Conexão estabelecida

Sending - Usuário está esperando permissão para transmitir o pacote

Receiving - Um RECEIVE foi feito.

Disconnecting - Um DISCONNECT foi feito localmente



## 6.3.2. Exemplo de Entidade de Transporte

- As transições de estados são feitas quando ocorrem chamadas de primitivas, ou chegada de pacotes, ou esgotamento do temporizador.
- Diferentemente desses processos que podem ser chamados pelo programador usuário, há também o `PacketArrival` e o `clock` que são rotinas de interrupções causadas por execuções externas, pela chegada de pacotes ou pelo tick do clock.



## 6.3.2. Exemplo de Entidade de Transporte

- Para um funcionamento apropriado, esses dois não devem ser executados enquanto a entidade de transporte estiver executando, ele deve ser executado só quando o processo usuário estiver em sleep ou estiver executando algo fora da entidade de transporte.
- A existência do bit Q (Qualifier) bit no cabeçalho de X.25 nos permite evitar um overhead de um cabeçalho de um protocolo de transporte. Normalmente são enviados mensagens com  $Q=0$ .



## 6.3.2. Exemplo de Entidade de Transporte

- Os protocolos de controle de mensagens que é só um (CREDIT) no nosso exemplo, são enviados os pacotes de dados com  $Q=1$ . Essas mensagens de controle são detectadas e processadas pela entidade de transporte.
- A estrutura de dados principal é o array conn, que tem uma gravação para cada potencial conexão. Essas gravações mantêm o estado de conexão incluindo o endereço das duas pontas, o número de mensagens enviadas e recebidas na conexão, o estado atual, o ponteiro do buffer usuário, o número de bytes da mensagem atual enviado ou recebido, um bit indicando que o usuário remoto fez um DISCONNECT, um temporizador, e um contador de permissão usado para habilitar as mensagens sendo enviadas.



## 6.3.2. Exemplo de Entidade de Transporte

- Quando um usuário fizer um CONNECT, a camada de rede enviará um CALL REQUEST para a máquina remota, e o usuário será colocado em sleep.
- Quando a CALL REQUEST chegar no outro lado, a entidade de transporte será interrompida para executar o PacketArrival. Se o usuário local estiver recebendo o endereço certo ele irá enviar o CALL ACCEPTED de volta e o usuário remoto acorda. Se não, ocorrerá o timeout.
- Se o LISTEN é feito dentro desse período, a conexão é efetuada com sucesso, senão é enviado um pacote CLEAR REQUEST. Isso serve para que o iniciador não seja bloqueado permanentemente.



## 6.3.2. Exemplo de Entidade de Transporte

- Apesar disso, nós ainda precisamos de uma forma de rastrear de volta qual pacote pertence a qual conexão.
- A forma mais simples é usando o número do circuito virtual de X.25 junto com o número de conexão.
- O número do circuito virtual pode ser usado como um índice em conn array. Com conexões iniciadas em um host, o número é escolhido pela entidade de transporte, para chamadas sendo recebidos a rede escolhe qualquer número de circuito virtual que não está sendo utilizado.



## 6.3.2. Exemplo de Entidade de Transporte

- Um mecanismo de controle de fluxo diferentemente da janela deslizante é usado, quando um usuário chama RECEIVE, uma mensagem de credit especial é enviada para a entidade de transporte na máquina que está enviando, e é guardada em conn array.
- Quando SEND é chamado, a entidade checa para ver se o credit chegou na conexão especificada. Se tiver chegado, a mensagem é enviada e o credit é decrementado, senão a entidade faz um sleep até que o credit chegue. Isso garante que nenhuma mensagem seja enviada a não ser que o outro lado efetue um RECEIVE.





### 6.3.3. Exemplo de Máquinas de Estados Finitos

- Escrever uma entidade de transporte é difícil, especialmente para as classes superiores de protocolo de transporte. Para reduzir a chance de cometer erros, é usualmente usado o estado do protocolo de estado de máquinas finitas.
- No nosso protocolo de exemplo há 7 estados por conexão. É também possível isolar 12 eventos que podem ocorrer ao mudar de um estado para outro. 5 deles são os serviços de primitivas, outros 6 são de chegada de tipos de pacotes. O último é o esgotamento do temporizador.



### 6.3.3. Exemplo de Máquinas de Estados Finitos

- A figura a ser mostrada a seguir mostra o protocolo principal agindo na forma de matriz. As colunas são estados e as linhas são os 12 eventos.
- Cada entrada na matriz tem 3 campos, um predicado, uma ação e um novo estado. O predicado indica sob quais condições a ação é feita. Por exemplo na entrada do canto superior esquerdo se um LISTEN é executado e não há mais espaço na mesa o LISTEN falha e o estado não muda.

		Status						
		Idle	Waiting	Queued	Established	Sending	Receiving	Disconnecting
Primitives	LISTEN	P1: ~/Idle P2: A1/Estab P2: A2/Idle		~/Estab				
	CONNECT	P1: ~/Idle P1: A3/Wait						
	DISCONNECT				P4: A5/Idle P4: A6/Disc			
	SEND				P5: A7/Estab P5: A8/Send			
	RECEIVE				A9/Receiving			
Incoming packets	CallReq	P3: A1/Estab P3: A4/Queu'd						
	CallAcc		~/Estab					
	ClearReq		~/Idle		A10/Estab	A10/Estab	A10/Estab	~/Idle
	ClearConf							~/Idle
	DataPkt						A12/Estab	
Clock	Credit				A11/Estab	A7/Estab		
	Timeout			~/Idle				

#### Predicates

P1: Connection table full  
P2: CallReq pending  
P3: LISTEN pending  
P4: ClearReq pending  
P5: Credit available

#### Actions

A1: Send CallAcc  
A2: Wait for CallReq  
A3: Send CallReq  
A4: Start timer  
A5: Send ClearConf  
A6: Send ClearReq  
A7: Send message  
A8: Wait for credit  
A9: Send credit  
A10: Set ClrReqReceived flag  
A11: Record credit  
A12: Accept message



### 6.3.3. Exemplo de Máquinas de Estados Finitos

- Mas se chegar um CALL REQUEST a conexão é estabelecida imediatamente. Outra Possibilidade é que P2 é falso, não há CALL REQUEST para chegar, nesse caso a conexão permanece IDLE esperando um CALL REQUEST.
- As ações de A1 até A12 são ações majoritárias, como mandar pacotes e iniciar temporizadores. Nem todas as ações minoritárias, como inicializar campos de conexão guardadas são atendidas. Se uma ação envolve acordar um processo em sleep, a ação seguinte também conta.



### 6.3.3. Exemplo de Máquinas de Estados Finitos

- As vantagens de representar o protocolo como uma matriz são três:
- Nessa forma é muito mais fácil para um programador checar cada combinação de estado e evento e ver qual a ação necessária.
- Na implementação, algumas combinações são usadas para tratar erros. No exemplo, se uma conexão está em estado de espera, o DISCONNECT é impossível, por causa o usuário está bloqueado e não consegue executar nenhuma primitiva.
- Em outra mão no estado SENDING, pacote de dados não são esperados porque nenhum credit houve problemas. A chegada do pacote de dados é um erro de protocolo que deve ser checado.



### 6.3.3. Exemplo de Máquinas de Estados Finitos

- A segunda vantagem está na implementação. Podemos usar arrays de duas dimensões  $a[i][j]$  sendo um ponteiro ou índice para o procedimento que trata a ocorrência do evento  $i$  quando está em um estado  $j$ .
- Uma possível implementação é escrever a entidade de transporte como um pequeno loop, esperando um evento no topo do loop. Quando um evento ocorre, a conexão é localizada e o estado extraído. Com o estado e o evento agora conhecidos, a entidade indexa no array  $a$  e chama o processo apropriado.



### 6.3.3. Exemplo de Máquinas de Estados Finitos

- A terceira vantagem do estado de máquina finito é para o protocolo de descrição. Em alguns documentos padrões, os protocolos são dados como um estado de máquina finito. Indo desse tipo de descrição para uma entidade de transporte já em funcionamento é muito mais simples se a entidade de transporte também é conduzido por um estado de máquina finito baseado nesse padrão.
- A desvantagem principal desse estado é que ele pode ser um mais difícil de entender que um exemplo de programação linear que usamos anteriormente. Esse problema pode ser facilmente solucionado desenhando o estado de máquinas finito como um grafo.



## 6.4. Exemplos de Camada de Transporte


- Nesta seção veremos alguns exemplos de camada de transporte, entre eles:
  1. Redes Públicas
  2. ARPANET
  3. MAP e TOP
  4. USENET








## 6.4.1. A Camada de Transporte em Redes Públicas

- Quase toda a rede pública utiliza o serviço de transporte OSI orientado a conexão (ISO 8072) e os protocolos de transporte OSI (ISO 8073).
- O protocolo de transporte OSI tem cinco variações: Classes 0 a 4.
- Cada variação destina-se a um tipo específico de confiabilidade da rede, variando perfeitamente para completamente não confiável.

- 
- O protocolo de transporte OSI possui 10 tipos diferentes de TPDU. Cada TPDU tem até quatro partes:
    1. Um campo de 1 byte que dá o comprimento do cabeçalho.
    2. Uma parte fixa do cabeçalho (o comprimento depende do tipo TPDU).
    3. Uma parte variável do cabeçalho (o comprimento depende dos parâmetros).
    4. Dados do usuário.

- 
- O primeiro byte de cada TPDU é o campo LI (Length Indicator).
  - Ele fornece o comprimento total do cabeçalho (fixo + partes variáveis) em bytes, excluindo o próprio campo LI, até um máximo de 254 bytes.
  - Em seguida, vem a parte fixa do cabeçalho. Esses campos são suficientemente importantes para serem incluídos em cada TPDU do tipo relevante.
  - Em outras palavras, a parte fixa de todas as requisições de conexão TPDU tem os mesmos campos, mas estes são diferentes dos campos em uma DATA TPDU.

- 
- Seguindo a parte fixa do cabeçalho está a parte variável.
  - Esta parte do cabeçalho é usada para opções que nem sempre são necessárias.
  - O destinatário de uma TPDU pode dizer quantos bytes de cabeçalho de parte variável estão presentes olhando o campo LI e subtraindo o comprimento do cabeçalho da parte fixa para o tipo TPDU.
  - A parte variável é dividida em campos, cada um começando com um campo de 1 byte de tipo, então um campo de 1 byte de comprimento, seguido do próprio dado.


- 
- Seguindo o cabeçalho, vem os dados do usuário.
  - O DATA TPDU contém dados do usuário.
  - Os formatos dos 10 tipos TPDU são mostrados na Figura 6-24 do próximo slide.


Bytes	1	1	2	2	1		
CR	LI	1110 Cdt	0...0	Source ref	Class option	Variable part	User data
CC	LI	1101 Cdt	Destination ref	Source ref	Class option	Variable part	User data
DR	LI	1000 0000	Destination ref	Source ref	Reason	Variable part	User data
DC	LI	1100 0000	Destination ref	Source ref	Variable part		
DT	LI	1111 0000	Destination ref	EOT TPDU N	User data		
ED	LI	0001 0000	Destination ref	EOT TPDU N	Variable part	User data	
AK	LI	0110 Cdt	Destination ref	TPDU expected	Variable part		
EA	LI	0010 0000	Destination ref	TPDU expected	Variable part		
RJ	LI	0101 Cdt	Destination ref	TPDU expected			
ER	LI	0111 0000	Destination ref	Reject cause	Variable part		

CR: Connection request  
 CC: Connection confirm  
 DR: Disconnect request  
 DC: Disconnect confirm  
 DT: Data


ED: Expedited data  
 AK: Data acknowledgement  
 EA: Expedited data acknowledgement  
 RJ: Reject  
 ER: Error


Fig. 6-24. The OSI transport protocol TPDU's.


- 
- CONNECTION REQUEST, CONNECTION CONFIRM, DISCONNECT REQUEST e DISCONNECT CONFIRM dos TPDUs são completamente análogas aos pacotes CALL REQUEST, CALL ACCEPTED, CLEAR REQUEST e CLEAR CONFIRM, usados no X.25.
  - Para estabelecer uma conexão, a entidade de transporte iniciante envia uma TPDU de Solicitação de Conexão e as respostas de pares com a Confirmação de Conexão.
  - Da mesma forma, para liberar uma conexão, qualquer uma das entidades de transporte envia uma Solicitação de Desconexão e as respostas dos pares com a Confirmação de Desconexão.


- 
- As TPDU's DATA e EXPEDITED DATA são usadas para dados regulares e acelerados, respectivamente.
  - Esses dois tipos são reconhecidos pelas TPDU's DATA ACKNOWLEDGMENT e EXPEDITED DATA ACKNOWLEDGMENT, respectivamente.
  - Finalmente, as TPDU's REJECT e ERROR são usadas para o tratamento de erros.




- 
- A CONNECTION REQUEST é usada para estabelecer uma conexão.
  - Como todas as TPDUs, ela contém um campo LI de 1 byte, dando o comprimento total do cabeçalho (excluindo o campo LI).
  - Em seguida, vem um byte contendo um tipo TPDU de 4 bits e o campo cdt (credit).
  - O protocolo de classe 4 usa um esquema de credit para o controle de fluxo, em vez de um esquema de janela deslizante, e este campo identifica a entidade de transporte remoto quanto TPDU pode inicialmente enviar.

- 
- Os campos de Referência de Destino e de Referência de Origem identificam conexões de transporte.
  - Eles são necessários porque nas Classes 2, 3 e 4 é possível multiplexar várias conexões de transporte em uma conexão de rede.
  - Quando um pacote vem da camada de rede, as entidades de transporte usam esses campos para determinar a qual conexão de transporte pertence a TPDU no pacote.
  - A CONNECTION REQUEST do TPDU fornece um identificador no campo de referência de Origem que será usado pelo iniciador.
  - A CONNECTION CONFIRM do TPDU adiciona a esse identificador a referência de Destino, que é usada pelo destino para a identificação da conexão.

- 
- O campo de opção Classe é usado pelas entidades de transporte para negociar a classe de protocolo a ser usada.
  - O iniciador faz uma proposta, que o respondente pode aceitar ou rejeitar.
  - O respondente também pode fazer uma contraproposta, sugerindo uma classe de protocolo inferior, mas não superior.
  - Este campo também contém dois bits que são usados para habilitar números de seqüência de TPDU de 4 bytes em vez dos números padrão de 1 byte e ativar ou desativar o controle de fluxo explícito na Classe 2.


- 
- A parte variável da REQUISIÇÃO DE CONEXÃO da TPDU pode conter uma das seguintes opções:
    1. TSAP para ser conectado ao host remoto
    2. TSAP está conectado ao host local.
    3. Tamanho TPOU máximo proposto (128 a 8192 bytes, em potências de 2).
    4. Número da versão.
    5. Parâmetro de proteção (por exemplo, uma chave de criptografia).
    6. Checksum.
    7. Alguns bits de opção (por exemplo, uso de dados acelerados, uso de checksum).


- 
8. Classes de protocolo alternativas que são aceitáveis para o iniciador.
  9. Máximo atraso antes de reconhecer uma TPDU, em milissegundos.
  10. Potência esperada (média desejada e mínima aceitável).
  11. Taxa de erro residual (média desejada e máxima aceitável).
  12. Prioridade (0 a 65535, sendo 0 a prioridade mais alta).
  13. Retardo de trânsito (médio e máximo aceitável, em milissegundos).
  14. Quanto tempo para continuar tentando recuperar após um N-RESET.



## 6.4.2. A Camada de Transporte na ARPANET (TCP)

- No design original da ARPANET, assumiu-se que uma sub rede ofereceria um serviço de circuito virtual, ou seja, que fosse perfeitamente confiável.
- O primeiro protocolo de camada de transporte foi o NCP (Network Control Protocol).
- Contudo, com o decorrer do tempo e o crescimento da empresa, foi-se necessário a criação de um novo protocolo de camada de transporte, o TCP (Transmission Control Protocol), associado ao protocolo de rede IP.

- 
- Uma entidade de transporte TCP aceita mensagens arbitrariamente longas do processos de usuário, as quebra em partes que não excedam os 64K bytes e envia cada parte como um datagrama separado.
  - A camada de rede não garante que os datagramas sejam entregues corretamente. Então, cabe ao TCP a tarefa de “time out” e retransmitir conforme necessário.
  - Datagramas podem, também, chegar fora de ordem e cabe ao TCP a tarefa de reorganizá-los na sequência adequada.

- 
- Cada byte de dados transmitidos pela TCP tem seu próprio número de seqüência particular.
  - O espaço do número de seqüência é de 32 bits de largura para garantir que as duplicatas antigas tenham desaparecido pelo tempo em que os números de seqüência se envolveram.
  - TCP lida explicitamente com o problema de duplicatas atrasadas.



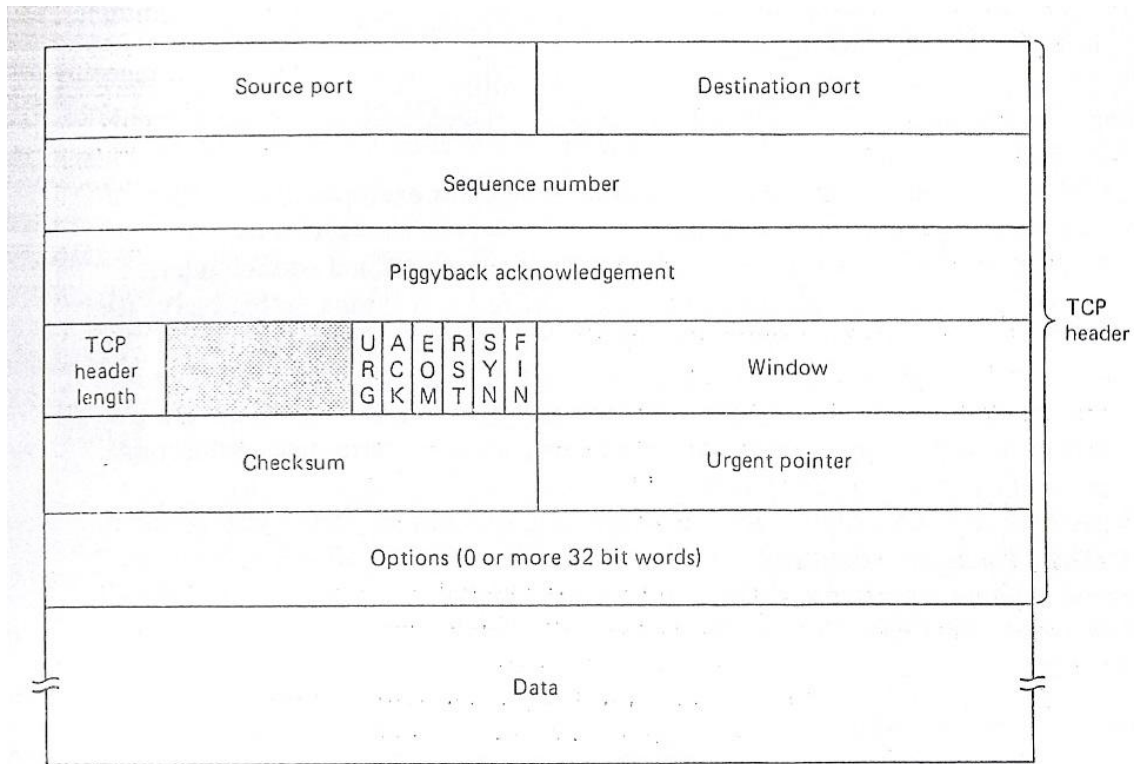





Fig. 6-26. The TCP TPDU structure.

- 
- O cabeçalho TCP mínimo é de 20 bytes.
  - Ao contrário do OSI Classe 4, com o qual é, aproximadamente comparável, TCP possui apenas um formato de cabeçalho TPDU.
  - A porta de Origem e os campos da porta de Destino identificam os pontos finais da conexão, ou seja, os endereços TSAP na terminologia OSI.
  - Cada host pode decidir por si próprio como alocar seus portos.


- 
- O número de sequência e os campos de confirmação de Piggyback executam suas funções usuais.
  - Eles têm 32 bits de comprimento porque cada byte de dados está numerado em TCP.
  - Em seguida vem seis flags de 1 bit (URG, ACK, EOM, RST, SYN, FIN).
  - O campo URG é definido como 1 se o ponteiro Urgent estiver em uso.
  - O ponteiro Urgent é usado para indicar um deslocamento de bytes do número de sequência atual em quais dados urgentes são encontrados.
  - O controle de fluxo no TCP é gerenciado usando uma janela de deslizamento de tamanho variável.

- 
- Um Checksum também é fornecido para extrema confiabilidade.
  - O algoritmo de checksum consiste simplesmente em adicionar todos os dados, considerados como palavras de 1-bit, e depois levar os complemento de 1 da soma.
  - O campo Opções é usado para coisas diversas, por exemplo para se comunicar com tamanhos de buffer durante o procedimento de configuração.



### 6.4.3. A Camada de Transporte no MAP e TOP

- MAP (Manufacturing Automation Protocol) e TOP (Technical Office Protocol) utilizam o protocolo de transporte OSI.
- Como utilizam um protocolo sem conexão, são obrigados a usar um variante TP4 na camada de transporte.
- TP4 contém várias opções que podem ser negociadas quando a conexão for estabelecida.

- 
- É necessário um computador que inicia a conexão para especificar o uso da Classe 4 no CR TPDU, assim como é necessário um computador que responda para que ele seja aceito na CC TPDU.
  - Se ambos os lados não forem capazes, ou não estiverem dispostos a fazer isso, a conexão não pode ser estabelecida.
  - Deve haver suporte para as opções de sequência de números de 7 e 31 bits.



## 6.4.4. Camada de Transporte no USENET

- USENET não possui um protocolo de transporte oficial.
- Cada par de máquinas de comunicação pode negociar o uso de qualquer protocolo de transporte desejado.
- As camadas superiores não exigem nenhum protocolo de transporte específico, embora precisem de uma maneira de estabelecer uma conexão confiável entre as máquinas.
- Se isso puder ser alcançado, cada parte máquinas pode usar qualquer protocolo de transporte mutuamente aceitável, ou nenhum deles, se a comunicação subjacente for suficientemente confiável.