



# Camada de Sessão do RM-OSI

Danilo Vieira França - [danilo11franca@yahoo.com](mailto:danilo11franca@yahoo.com)

Matheus Henrique Ferreira Protásio - [mathzhf@gmail.com](mailto:mathzhf@gmail.com)

Marcos Gabriel Leão Muñoz - [marcosgmunoz@gmail.com](mailto:marcosgmunoz@gmail.com)

João Vitor Rodrigues de Alcântara Ferreira [joao-vitor-alcantara@hotmail.com](mailto:joao-vitor-alcantara@hotmail.com)

---

# Índice

Introdução

7.1 - Serviços Oferecidos pela Camada de Sessão

7.1.1 - Serviços Fornecidos para a Camada de Apresentação

7.1.2 - Troca de Dados

7.1.3 - Gerenciamento de Diálogos

7.1.4 - Sincronização

7.1.5 - Gerenciamento de Atividade

7.1.6 - Relatório de Exceções

7.1.7 - Primitivas do Serviço de Sessão

7.2 RPC - Remote procedure call

7.3 - Exemplos da Camada de Sessão

7.3.1 - Camada de Sessão em Redes Públicas

7.4 - Sumário



## Introdução

7-Aplicação

6-Apresentação

5-Sessão

4-Transporte

3-Rede

2-Enlace de Dados

1-Física

O Modelo OSI é composto por 7 camadas, essas camadas são divididas em camadas superiores e camadas inferiores.

A camada de sessão é uma das 3 camadas consideradas de mais “alto nível” do modelo OSI (camadas superiores), junto com as camadas de Aplicação e Apresentação.

Em contraste com as 4 camadas inferiores que tem foco em prover uma comunicação confiável de ponta a ponta, as camadas superiores têm como preocupação fornecer serviços orientados para o usuário.



As camadas superiores utilizam o canal livre de erros provido pela camada de transporte, e recursos adicionais que são úteis para várias aplicações, para que assim os criadores dessas aplicações não tenham que implementar esses recursos novamente como parte de um programa separado.

Esta camada foi criada pela ISO (International Organization for Standardization) , não sendo encontrada em redes de computadores que antecedem este modelo.

A camada de sessão é considerada uma camada “fina” se comparada a camadas inferiores, possuindo assim menos recursos.



A Camada de Sessão é responsável pelos processos que controlam a transferência dos dados, cuida dos erros e administra os registros das transmissões.

Nela são implementadas regras para sincronização das trocas de mensagens, e por averiguar quais procedimentos a serem tomados em caso de falhas.

A camada de sessão tem como objetivo o fornecimento desse serviço de gerência de conexão entre aplicações, provendo ainda mecanismos de segurança, autenticação e sincronismo entre as partes. Podemos dizer que ela permite a comunicação de ponta a ponta entre aplicações em máquinas remotas, provendo ainda segurança via mecanismo de criptografia e gerência de sessão.



## 7.1 Serviços Oferecidos pela Camada de Sessão

**Troca de Dados** - estabelecer conexão com outro usuário, trocar dados e fechar a conexão;

**Gerenciamento de Diálogos** - negociar a utilização de tokens para troca de dados, sincronização e liberação da conexão de sessão;

**Sincronização** - definir pontos de sincronização em diálogos possibilitando interrupções e retornos (caso ocorram erros, o diálogo deve ser retomado a partir do ponto de sincronização);

**Gerenciamento de Atividades** - permite que mensagens sejam divididas pelo usuário em unidades lógicas menores independentes (atividades);

**Relatório de Exceções** - caso ocorram problemas, estes podem ser relatados ao parceiro de um determinado usuário.



### **7.1.1 Serviços Fornecidos para a Camada de Apresentação**

Siglas:

SSAP - Session Service Access Point

SPDU - Session Protocol Data Unit

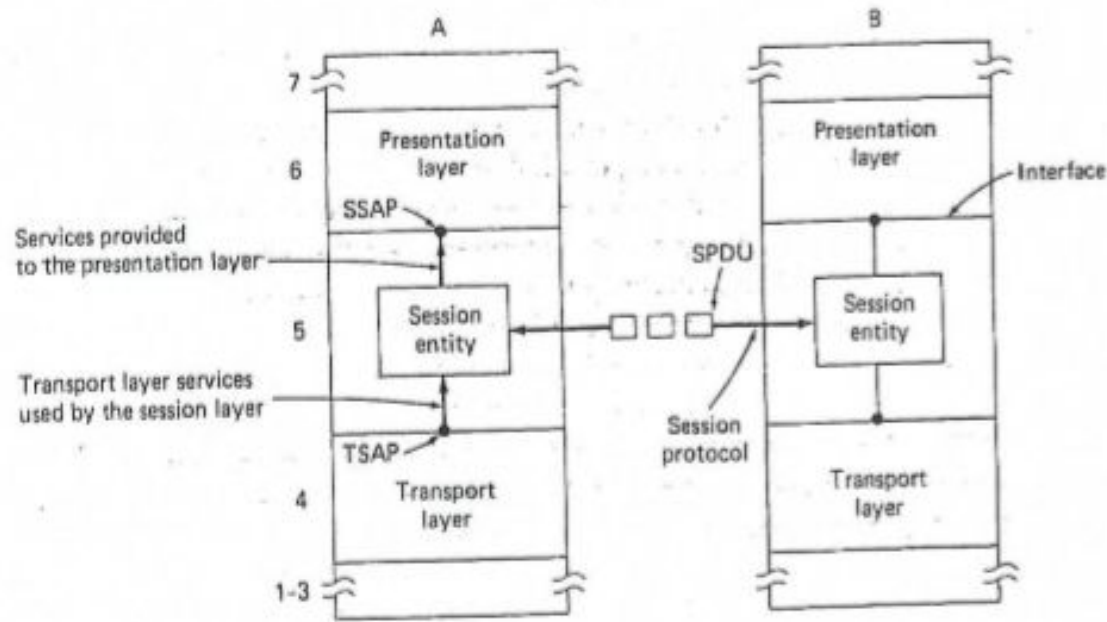


Fig. 7-1. The transport, session, and presentation layers.

A camada de sessão deve fornecer um caminho para os usuários estabelecerem conexões, estas chamadas **sessões**, e transferirem dados nelas de forma ordenada.

Uma sessão pode ser usada para transferência de um arquivo, acesso remoto de um terminal à um computador distante, etc.



Uma sessão tem algumas semelhanças com uma conexão de transporte, mas elas não são idênticas. Geralmente quando um *request* chega até a camada de sessão para estabelecer uma sessão, a conexão de transporte deve estar estabelecida, para que assim ela leve a conexão.

Quando a sessão acabar, a conexão de transporte é liberada.

No caso mencionado anteriormente temos um exemplo de mapeamento um para um entre a sessão e a conexão de transporte. No entanto, existem outros tipos de mapeamento.

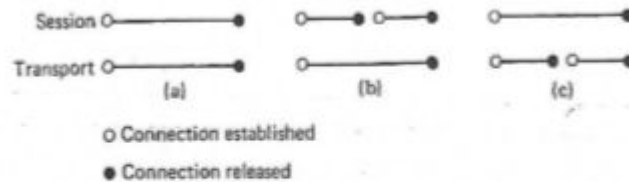



Fig. 7-2. Three ways of mapping sessions onto transport connections. (a) One-to-one mapping. (b) Consecutive sessions use the same transport connection. (c) One session spans multiple transport connections. The horizontal axis is time.



Não é permitido multiplexar várias sessões em uma única conexão de transporte, como ocorre na camada de transporte, quando esta consegue multiplexar várias conexões de transporte em uma conexão de rede.

Cada conexão de transporte carrega no máximo uma sessão por vez.

A multiplexação é feita para reduzir custos ou melhorar o desempenho, o que são funções da camada de transporte.




### 7.1.2 Troca de Dados

O recurso mais importante da Camada de Sessão é a troca de dados. Uma sessão, assim como uma conexão de transporte, passa por três fases: estabelecimento, utilização e liberação.

O estabelecimento de sessão é feito através de um pedido de conexão à camada de transporte, e envolve a negociação entre usuários no que tange aos diversos parâmetros da conexão. Muitas primitivas são pertinentes à conexão de transporte e são simplesmente passadas para esta conexão sem qualquer modificação

Na maioria dos casos, o que a entidade de sessão tem que fazer é invocar a primitiva de transporte correspondente à primitiva invocada pelo usuário da sessão, para que assim o trabalho seja feito.

Ex.: O usuário da sessão manda S-CONNECT.request para estabelecer a sessão, o provedor da sessão manda T-CONNECT.request para estabelecer a conexão de transporte.



Apesar das similaridades nas primitivas, a liberação das sessões e das conexões de transporte é feita de maneira distinta.

Conexões de transporte são terminadas com a primitiva `T-DISCONNECT.request` o que gera uma liberação abrupta e pode causar perda de dados que estavam sendo trocados no momento da liberação.

Sessões são finalizadas com a primitiva `S-RELEASE.request` que gera uma liberação ordenada, onde não há perda de dados. As sessões têm um equivalente a liberação abrupta com a primitiva `S-U-ABORT.request`.

A camada de sessão pode transmitir quatro tipos diferentes de dados: regulares e expedidos (análogos aos tipos da camada de transporte), tipificados e dados de capacidade.

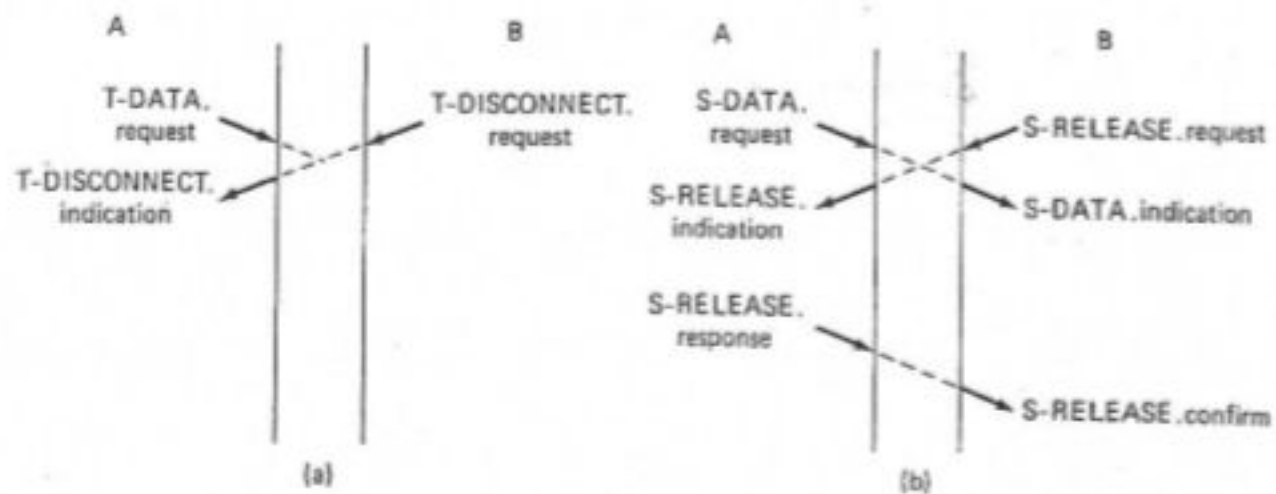


Fig. 7-3. (a) Abrupt release. (b) Orderly release.



### 7.1.3 Gerenciamento de Diálogos

A princípio, todas as conexões do Modelo OSI são full duplex. No entanto, existem situações em que o software da camada superior é estruturado para esperar que os usuários se revezem (comunicação half-duplex). Assim, foram introduzidos controles para determinar de quem é a vez de transmitir.

O Gerenciamento de Diálogos foi implementado através do uso de Tokens.

Quando a sessão é estabelecida, half-duplex é uma das opções. Se a opção half-duplex é selecionada, no primeiro diálogo estabelecido, é selecionado qual lado terá o Token primeiro.

Somente o usuário que está com o Token pode transmitir dados, o outro aguarda sua vez.

Quando a transmissão é encerrada, o Token é passado para o outro usuário. (usando S-TOKEN-GIVE.request).

Se o usuário que não está com o Token quer transmitir dados, ele pode pedir por meio da primitiva S-TOKEN-PLEASE.request, esse pedido pode ser aceito ou não pelo outro usuário. Se a opção full duplex for selecionada inicialmente, não há uso de Tokens para estabelecer o diálogo.

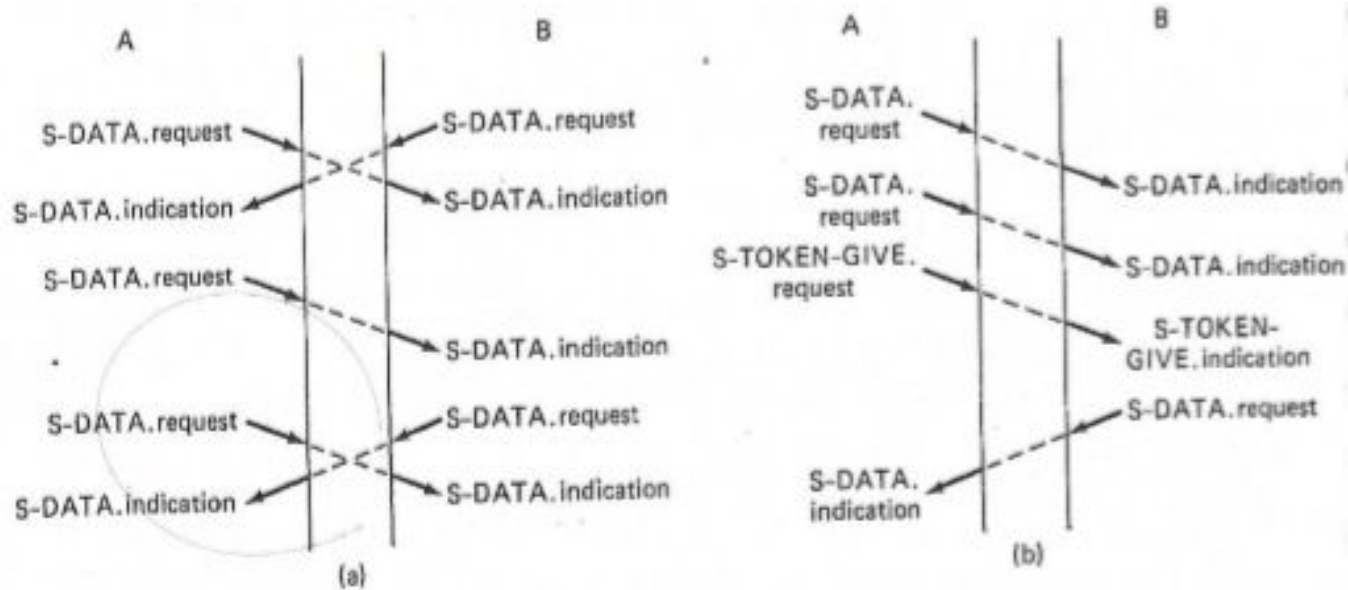


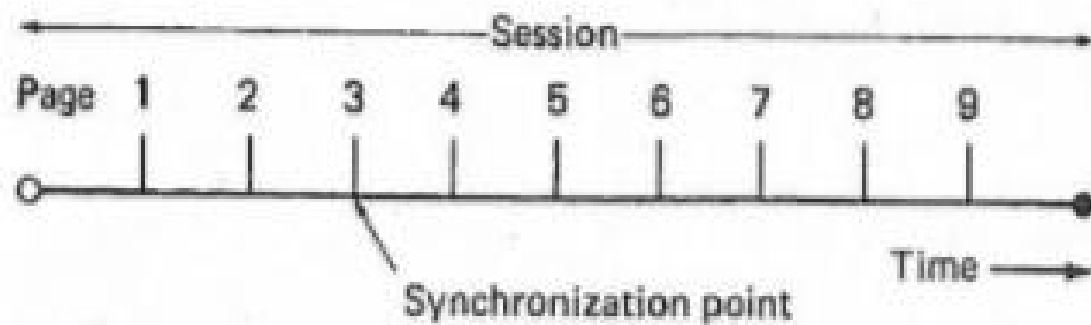
Fig. 7-4. (a) Full-duplex communication. (b) Half-duplex communication managed with a token.



### 7.1.4 Sincronização

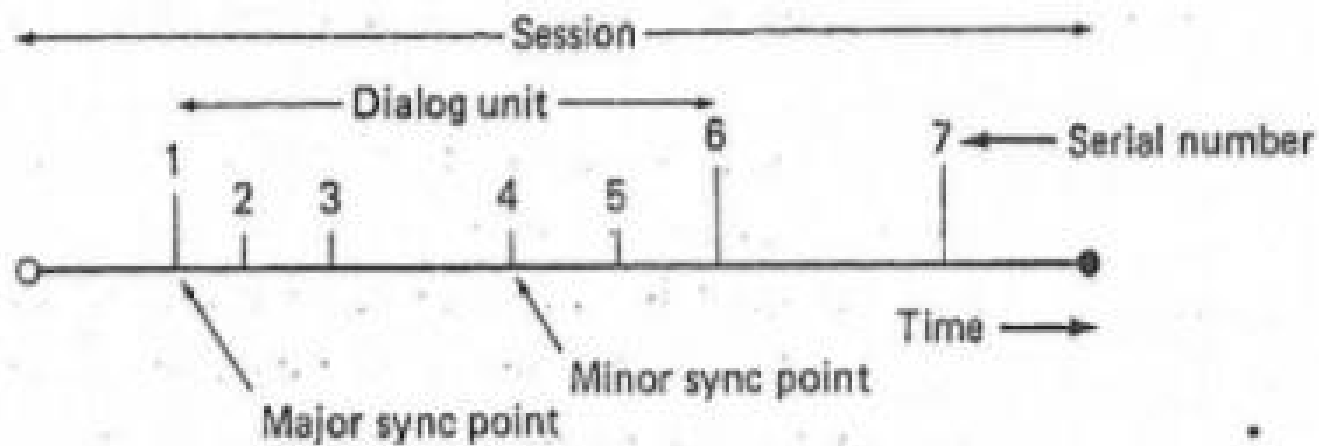
A sincronização é responsável por devolver as entidades na camada de sessão à um estado conhecido, devido a erros ou divergências.





**Fig. 7-5. Synchronization points.**

- O usuário pode inserir os pontos de sincronização.
- Cada ponto de sincronização vem com um número de série.
- Quando um usuário faz um requerimento de sincronização, o outro recebe uma indicação disso, o mesmo acontece para a ressincronização.
- O armazenamento e retransmissão de mensagens é de responsabilidade das camadas superiores.



**Fig. 7-6. Major and minor synchronization points.**

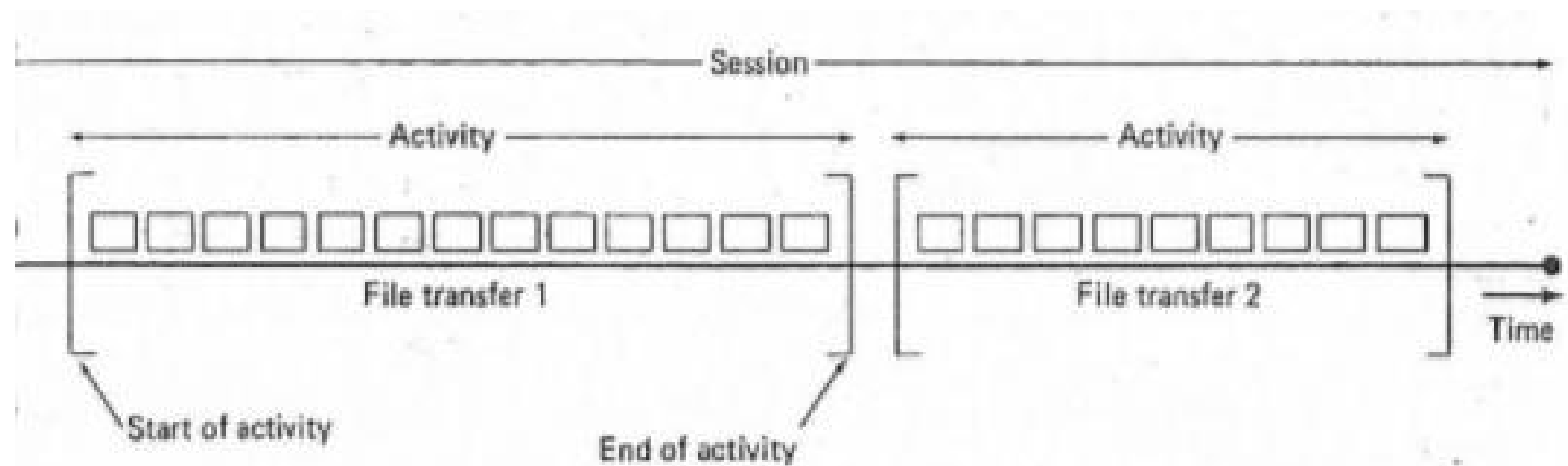


## 7.1.5 Gerenciamento de Atividade

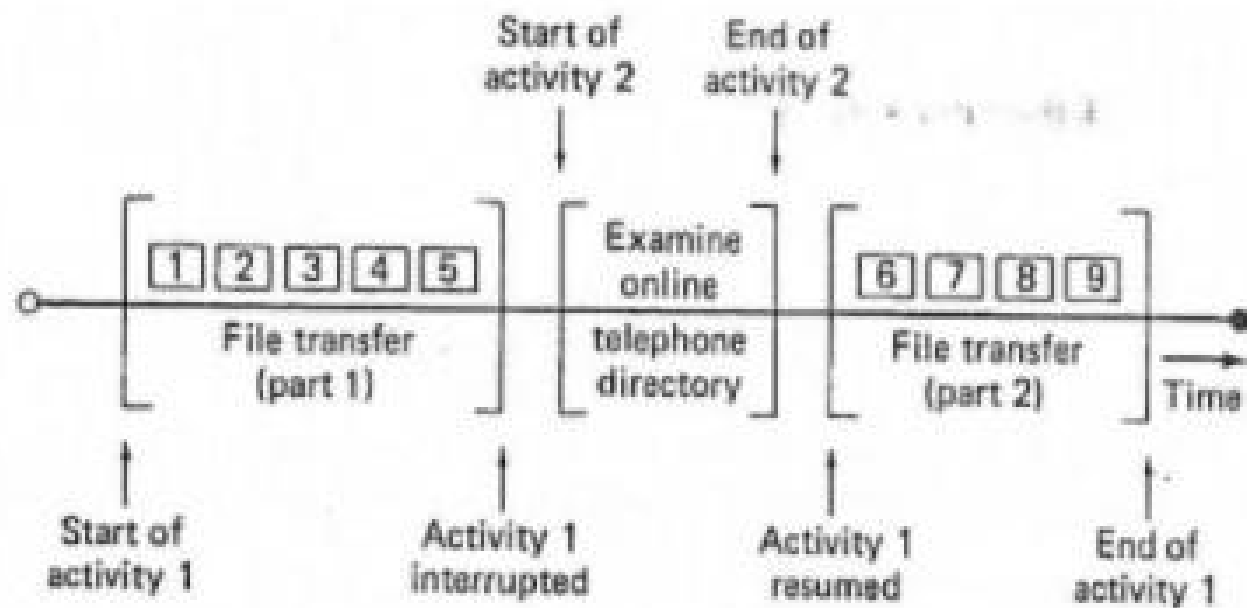
Também relacionado com sincronização, o gerenciamento de atividade permite que o usuário possa dividir a mensagem em unidades lógicas chamadas atividades.

### Primitivas usadas:

- S-ACTIVITY-START
- S-ACTIVITY-END
- S-ACTIVITY-DISCARD
- S-ACTIVITY-INTERRUPT
- S-ACTIVITY-RESUME



**Fig. 7-7.** Use of activities to mark file boundaries.



**Fig. 7-8.** Activities can be interrupted and later resumed.

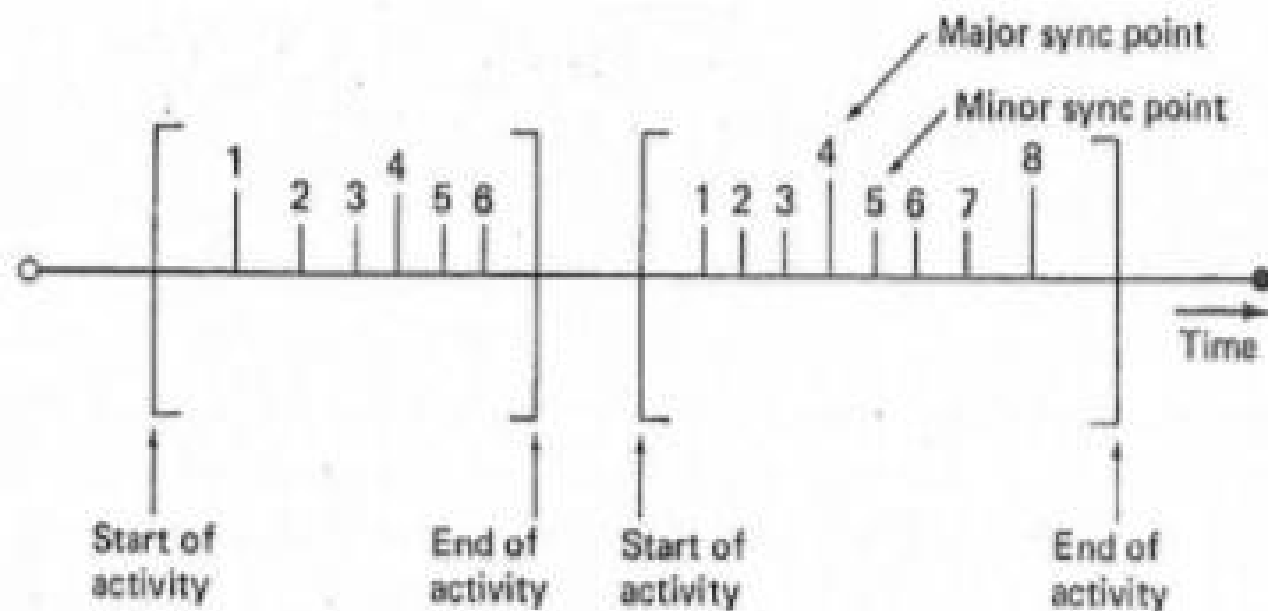


Fig. 7-9. Activities, major synchronization points, and minor synchronization points.



### 7.1.6 Relatório de exceções

Mecanismo de Propósito geral para reporte de erros inesperados. O usuário envia uma primitiva `S-U-EXCEPTION-REPORT.request` para o outro par informando do problema no campo de dados.

O provedor do serviço também pode usar dessa ferramenta para notificar o usuário de erros internos, erros da camada de transporte ou das camadas inferiores.

Porém, a decisão do que fazer é sempre do usuário.





### 7.1.7 Primitivas do serviço de sessão

As primitivas do serviço da camada de sessão podem ser divididos em 7 grupos

1. Estabelecimento de conexão
2. Encerramento de conexão
3. Transferência de dados
4. Gerenciamento de tokens
5. Sincronização
6. Gerenciamento de atividade
7. Relatório de exceções

OSI session primitive	Request	Indication	Response	Confirm	Meaning
S-CONNECT	X	X	X	X	Establish a session
S-RELEASE	X	X	X	X	Terminate a session gracefully
S-U-ABORT	X	X			User-initiated abrupt release
S-P-ABORT		X			Provider-initiated abrupt release
S-DATA	X	X			Normal data transfer
S-EXPEDITED-DATA	X	X			Expedited data transfer
S-TYPED-DATA	X	X			Out-of-band data transfer
S-CAPABILITY-DATA	X	X	X	X	Control information data transfer
S-TOKEN-GIVE	X	X			Give a token to the peer
S-TOKEN-PLEASE	X	X			Request a token from the peer
S-CONTROL-GIVE	X	X			Give all the tokens to the peer
S-SYNC-MAJOR	X	X	X	X	Insert a major sync point
S-SYNC-MINOR	X	X	X	X	Insert a minor sync point
S-RESYNCHRONIZE	X	X	X	X	Go back to a previous sync point
S-ACTIVITY-START	X	X			Start an activity
S-ACTIVITY-END	X	X	X	X	End an activity
S-ACTIVITY-DISCARD	X	X	X	X	Abandon an activity
S-ACTIVITY-INTERRUPT	X	X	X	X	Suspend an activity
S-ACTIVITY-RESUME	X	X			Restart a suspended activity
S-U-EXCEPTION-REPORT	X	X			Report of a user exception
S-P-EXCEPTION-REPORT		X			Report of a provider exception

(a)

S-UNITDATA	X	X			Connectionless data transfer
------------	---	---	--	--	------------------------------


(b)

Fig. 7-10. (a) OSI connection-oriented session service primitives. (b) OSI connectionless session service primitives.



## 7.2 RPC - Remote procedure call

A Chamada de Procedimento Remoto(RPC) é um modelo de sessão sem conexão.



-Desenvolvido como um modelo de diálogo e controle de erro radicalmente diferente do Modelo OSI.

-Feito para ser rápido, não contém uma estrutura com várias camadas.

## 7.2.1 Modelo Cliente-Servidor

-Para o exemplo serão usados computadores menores sem disco e servidores de arquivos que contém todos os dados, se comunicando por uma rede.

-Os clientes acessam os dados enviando pedidos aos servidores, que respondem com os dados pedidos. Comunicação sempre é iniciada pelo cliente.

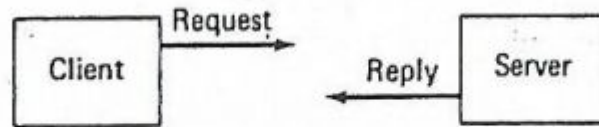



Fig. 7-12. In the client-server model, the client sends a request and the server sends a reply.



-É possível estabelecer sessões entre clientes e servidores usando uma comunicação com *half-duplex*, mas, especialmente em LANs, a comunicação desconectada é uma alternativa mais atraente.

-A diferença entre o Modelo RPC e os outros modelos com comunicação desconectada é que no RPC a comunicação entre computadores distintos funciona similarmente à comunicação local de uma máquina, omitindo a separação das máquinas de ambas as partes.



-É possível aumentar essa transparência incorporando RPC na própria linguagem de programação, usando comandos prontos que agirão nas duas máquinas automaticamente.

-Ex.: `read(file, buffer, count)`

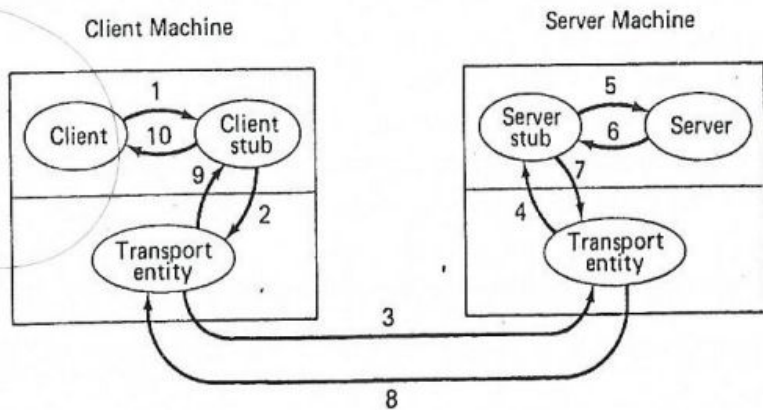
-Dessa forma, comunicação cliente-servidor toma a forma de chamadas de procedimentos, dispensando comandos de I/O ou interrupções.

-Esses procedimentos são chamados de *stubs*.

-Também pode pedir execuções arbitrárias no próprio servidor de forma muito mais intuitiva do que usando os comandos de I/O e interrupções tradicionais.

-Ex.: `delete(file)`

## 7.2.2 Implementação do RPC




- 1-Cliente chama um procedimento(stub);
- 2-Passa uma mensagem com parâmetros para a sua entidade de transporte;
- 3-A mensagem simplesmente recebe um header e é enviada para a rede;
- 4-Entidade de transporte do servidor passa a mensagem para seu stub;
- 5-Stub chama o procedimento do servidor;
- 6-Após processar o requisito, devolve resultado para um stub se for o caso;
- 7, 8, 9, 10- Faz o caminho inverso até devolver o resultado para o cliente.





## 7.2.3 Dificuldades na transparência

- Transferir valores é fácil, mas se o procedimento usa referências como parâmetros, não é possível ser passado por RPC, visto que os ponteiros serão inúteis se usados na memória de outro sistema.
- Uma solução seria fazer com que o procedimento stub copie os dados dos ponteiros para serem enviados nas mensagens. Porém, há certas situações em que isso não irá funcionar.



```
program test(output);  
var a:integer;  
  
procedure doubleincr(var x,y:integer);  
begin  
  x := x+1;  
  y := y+1  
end;  
  
begin                                {main program}  
  a := 0;  
  doubleincr(a, a);  
  writeln(a)  
end.
```

Fig. 7-14. If the procedure *doubleincr* is run remote, the program fails.

-Quando executado remotamente, o procedimento *doubleincr*(a, a) irá enviar duas cópias de 'a' e incrementá-las separadamente.

-Alguns sistemas RPC proíbem o uso de referência ou ponteiros nos stubs. Apesar de resolver o problema, há uma quebra na transparência do sistema, já que a regra para procedimentos locais não é mais a mesma dos procedimentos remotos.

# Como o stub do cliente sabe qual servidor chamar?

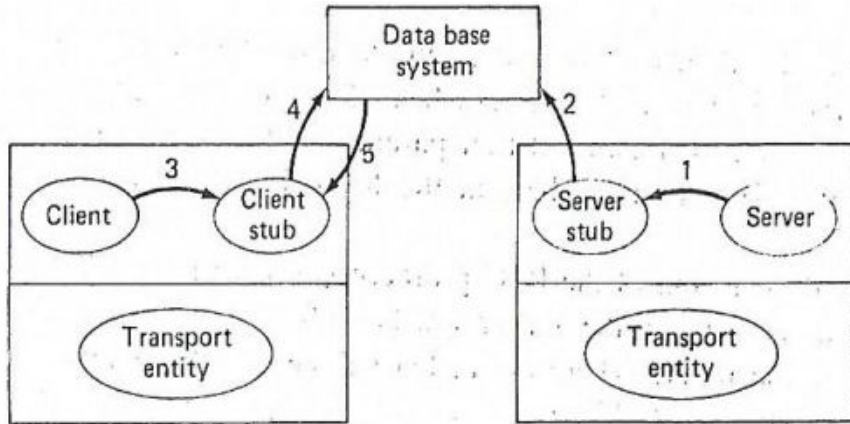



Fig. 7-15. Client-server binding is done via a data base.

- RPC usa um sistema mais simples e dinâmico.
- Cada servidor se registra no banco de dados após sua criação.
- Quando o cliente vai fazer sua primeira chamada ao servidor, o stub usa o nome do servidor para obter do banco de dados o endereço de rede e identificador do servidor.
- O processo (binding) é feito somente na primeira chamada. Após ele, o cliente sabe encontrar o servidor.



-Chamadas também precisam de um identificador de transação para evitar duplicações. Pode ser usado um timer, que ao ser esgotado desencadeia um reenvio. Se o servidor detecta o mesmo identificador de transação duas ou mais vezes, pode ignorar o excesso.

-Ao contrário de procedimentos locais, muitas coisas podem dar errado em RPCs (servidor caído, falha na rede, etc).

-Tratamento de exceção pode variar dependendo da linguagem usada, mas em geral é preferível que seja tratada por um *exception handler* ao invés de retornar o controle para quem requisitou a chamada. Mesmo sendo altamente variável, tratamento de exceção claramente é necessário para diferenciar chamadas bem-sucedidas de mal-sucedidas.



# Server crash: Como lidar com o problema

-O que fazer se o servidor sofrer um crash logo após processar um pedido e antes de enviar a resposta?

-Três alternativas mais relevantes:

1. Esperar uma resposta para sempre.
2. Levantar uma exceção ou reportar o erro para o cliente após o timeout.
3. Retransmitir o requisito após o timeout.

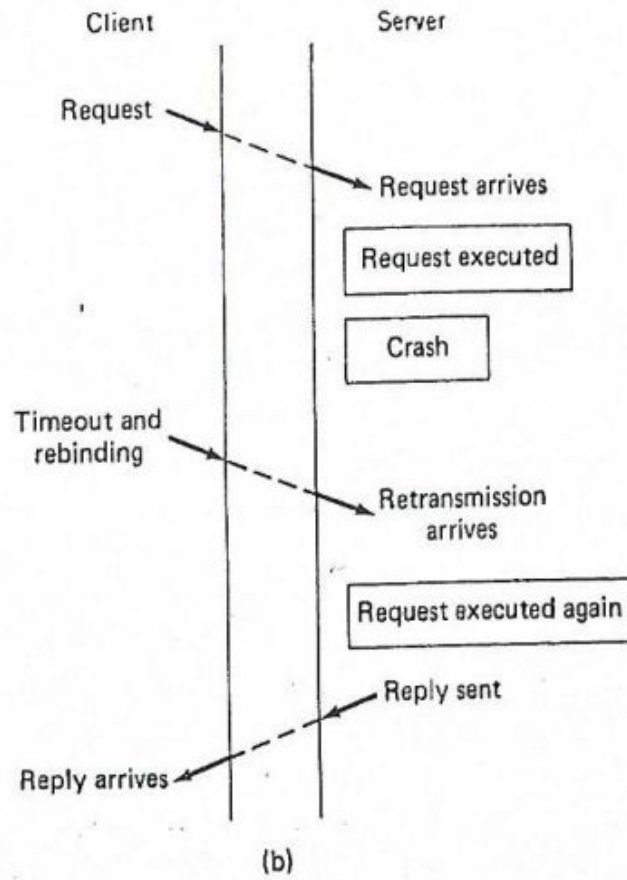
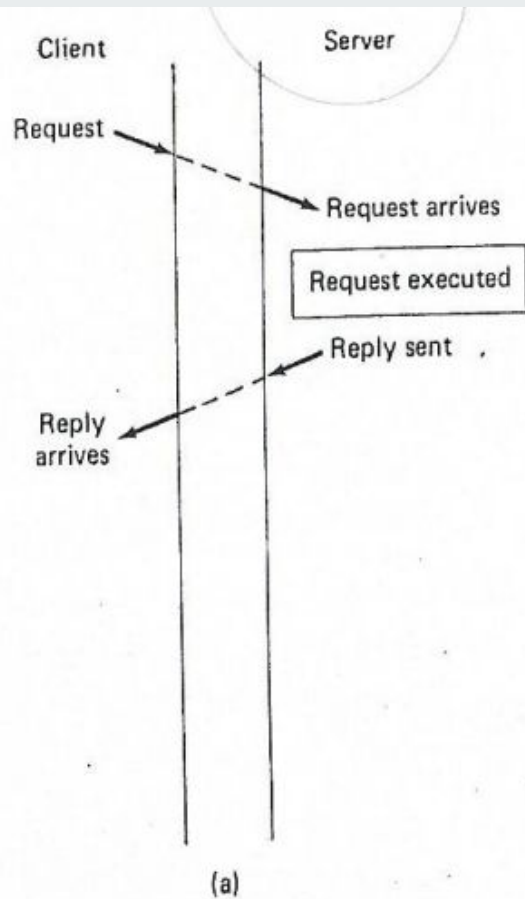


Fig. 7-16. (a) Ordinary two-message RPC. (b) RPC with a server crash and timeout by the client.



-Se a execução duplicada é aceitável ou não vai depender da operação.

-Idealmente: Execução *exactly once*. Não se conhece um método para ser alcançado.

-Com exceções é possível identificar um crash, talvez executar um rollback. Nesse caso a programação vai estar no exception handler, não no cliente.

-Alternativas: *at most once* e *at least once*.


-*At least once*: ideal para operações idempotentes. Caso contrário, pode usar um identificador de transação para cada tentativa, e o servidor pode identificar qual executar.



# Servidores órfãos

- Nome dado quando um procedimento continua executando mesmo se o cliente que o chamou (pai) sofreu um crash.
- Nelson (1981) descreveu 4 jeitos de lidar com esses processos.





-Exterminação: stubs do cliente criam logs antes de executar uma RPC. No caso de crash, esse log é utilizado para, após reinicialização, requisitar que cada servidor com processo ativo o encerre.

-Expiração: O servidor é dado uma quantia de tempo para completar a chamada. Se o tempo esgotar, ele pede uma outra quantia para o cliente. Nesse evento, ele detecta um possível crash e se desliga sozinho.

-Reencarnação: usado se o cliente não consegue exterminar todos os órfãos (particionamento na rede, por exemplo). O cliente envia o começo de uma nova “época” para todas as máquinas, que incorporarão essa informação nas suas respostas. Nesse caso, um processo que não recebeu a atualização terá sua resposta com a época errada, portanto poderá ser ignorado.

-Reencarnação delicada: A diferença é que nesse método, os servidores procuram o cliente quando uma nova época é declarada. Caso não o encontre, aí sim encerra o processo.



# Design de sistemas RPC

-4 categorias principais:

1. Design de interface
2. Design de cliente
3. Design de servidor
4. Design de protocolo



# Interface

-É preciso ter consciência de até onde transparência será uma prioridade, com problemas com ponteiros e a impossibilidade de alcançar execuções *exactly one* no caso de crashes. Alguns autores defendem que transparência nem deve ser tentada, sendo melhor deixar claro para o desenvolvedor o tipo de sistema com que está lidando.

-Por outro lado, mesmo que um sistema RCP pareça dar muita dor de cabeça no papel, na realidade crashes são raros, e a maioria dos outros problemas podem ser evitados com um design cauteloso de stubs e uma linguagem bem adequada.

-Stubs, tratamento de exceção e binding são pontos que precisam de atenção especial nessa parte de um projeto.



# Cliente

-Os pontos principais do design são os timeouts e os órfãos, mais precisamente como o cliente irá lidar com os erros, tendo em mente as alternativas discutidas anteriormente.



# Servidor

-A questão mais relevante é o paralelismo, se os servidores serão capazes de criar vários processos para lidar com vários requisitos ou se os requisitos entrarão em fila para execução única de cada processo do servidor.

-A primeira alternativa coloca a carga da criação de processos nas RPCs, mas dá a possibilidade de paralelismo, enquanto a segunda opção é mais simples e rápida mas não permite paralelismo.



# Protocolo

-A maior preocupação com o protocolo é atingir um desempenho alto. Nesse ponto, as soluções são muitas e variadas demais para discutir nessa apresentação, variando da implementação de RPCs na camada de apresentação até o uso puro de RPCs, praticamente por cima de camadas de transporte e rede nulas.



## 7.3 Exemplos da Camada de Sessão

Nesta parte falaremos um pouco como é usada a Camada de Sessão em Redes Públicas.



### 7.3.1 Camada de Sessão em Redes Públicas

Redes Públicas e outras redes OSI geralmente implementam os serviços completos da camada de sessão como descrito anteriormente neste capítulo. O protocolo de sessão é complexo devido ao número de serviços primitivos porém é seguido a risca. Esta complexidade vem de uma decisão para compatibilidade do antigo protocolo teletex CCITT.

Para cada primitiva de serviço, existe uma SPDU(Session Protocol Data Unit) que é enviada quando o protocolo é convocado. As primitivas que possuem response, é gerada outra SPDU para esta response.


Podemos observar a lista de serviços das primitivas abaixo e suas SPDUs abaixo:



OSI session primitive	Request Indication Response Confirm				SPDU sent on request	SPDU sent on response
	X	X	X	X		
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE	X	X	X	X	FINISH	DISCONNECT, NOT FINISHED (ABORT ACCEPT)
S-U-ABORT	X	X			ABORT	
S-P-ABORT		X			(ABORT)	
S-DATA	X	X			DATA TRANSFER	CAPABILITY DATA ACK
S-EXPEDITED-DATA	X	X			EXPEDITED DATA	
S-TYPED-DATA	X	X			TYPED DATA	
S-CAPABILITY-DATA	X	X	X	X	CAPABILITY DATA	
S-TOKEN-GIVE	X	X			GIVE TOKENS	(GIVE TOKENS ACK)
S-TOKEN-PLEASE	X	X			PLEASE TOKENS	
S-CONTROL-GIVE	X	X			GIVE TOKENS CONFIRM	
S-SYNC-MAJOR	X	X	X	X	MAJOR SYNC POINT	MAJOR SYNC POINT ACK
S-SYNC-MINOR	X	X	X	X	MINOR SYNC POINT	MINOR SYNC POINT ACK
S-RESYNCHRONIZE	X	X	X	X	RESYNCHRONIZE	RESYNCHRONIZE ACK
S-ACTIVITY-START	X	X			ACTIVITY START	ACTIVITY END ACK ACTIVITY DISCARD ACK ACTIVITY INTERRUPT ACK
S-ACTIVITY-END	X	X	X	X	ACTIVITY END	
S-ACTIVITY-DISCARD	X	X	X	X	ACTIVITY DISCARD	
S-ACTIVITY-INTERRUPT	X	X	X	X	ACTIVITY INTERRUPT	
S-ACTIVITY-RESUME	X	X			ACTIVITY RESUME	
S-U-EXCEPTION-REPORT	X	X			EXCEPTION REPORT	
S-P-EXCEPTION-REPORT		X			(EXCEPTION REPORT)	

Usando como exemplo de funcionamento, as primitivas CONNECT, RELEASE e ABORT :


OSI session primitive	Request	Indication	Response	Confirm	SPDU sent on request	SPDU sent on response
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE	X	X	X	X	FINISH	DISCONNECT, NOT FINISHED
S-U-ABORT	X	X			ABORT	(ABORT ACCEPT)
S-P-ABORT		X			(ABORT)	



OSI session primitive	Request	Indication	Response	Confirm	SPDU sent on request	SPDU sent on response
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE	X	X	X	X	FINISH	DISCONNECT, NOT FINISHED
S-U-ABORT	X	X			ABORT	(ABORT ACCEPT)
S-P-ABORT		X			(ABORT)	

## S-CONNECT:


Quando chamada, a entidade de sessão constrói um CONNECT SPDU e envia ao destino. Quando este SPDU chega na máquina remota, a entidade de sessão aqui causa um S-CONNECT.indication para acontecer. O usuário endereçado pode aceitar ou recusar a sessão convocando um S-CONNECT.response com parâmetros apropriados de aceitação ou recusa. O resultado dessa ação resultada na entidade remota de sessão mandando um ACCEPT ou REFUSE SPDU de volta para a máquina de origem.



OSI session primitive	Request	Indication	Response	Confirm	SPDU sent on request	SPDU sent on response
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE	X	X	X	X	FINISH	DISCONNECT, NOT FINISHED
S-U-ABORT	X	X			ABORT	(ABORT ACCEPT)
S-P-ABORT		X			(ABORT)	

## S-RELEASE:

Assim como a S-CONNECT, S-RELEASE também tem duas variações. Quando o usuário requer o fim de um sessão, um FINISH SPDU é transmitido. O usuário remoto pode também aceitar o pedido para terminar ou rejeitá-lo, no caso a sessão continua. A escolha é indicada por um parâmetro na primitiva S-RELEASE.response. Se o usuário tenha decidido aceitar o pedido e encerrar a sessão, a resposta ao FINISH é DISCONNECT. Se, entretanto, a oferta é recusada, o NOT FINISHED SPDU é mandado de volta e a sessão continua como se nada tivesse ocorrido.



OSI session primitive	Request	Indication	Response	Confirm	SPDU sent on request	SPDU sent on response
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE	X	X	X	X	FINISH	DISCONNECT, NOT FINISHED
S-U-ABORT	X	X			ABORT	(ABORT ACCEPT)
S-P-ABORT		X			(ABORT)	

### S-U-ABORT:

O padrão de uso de uma SPDU para S-U-ABORT difere do padrão normal. Contudo não há resposta possível para esta primitiva, o ABORT SPDU enviado quando a primitiva chamada é explicitamente reconhecida por ABORT ACCEPT pois ela não é gerada pela primitiva response, mas sim pela entidade de sessão própria. A razão para este acknowledgement é garantir que a conexão esteja livre de SPDUs, assim liberando o transporte de conexão para ser usado imediatamente por outra sessão.

## Formato das PDUs

O formato geral de uma PDU está mostrado ao lado. O campo SI(Session Identifier) consiste em 1 byte que dá o tipo da SPDU, as quais são distintas. O campo LI(Length Identifier) é um valor entre 0 e 254 dizendo quantos bytes de parâmetro acompanham. Caso passe de 254, LI recebe o valor de 255 e 2 bytes adicionais seguidos, até um máximo de 65,535 bytes.

(a) Formato geral de uma SPDU (b)-(d) Exemplo de Parâmetros de Campo

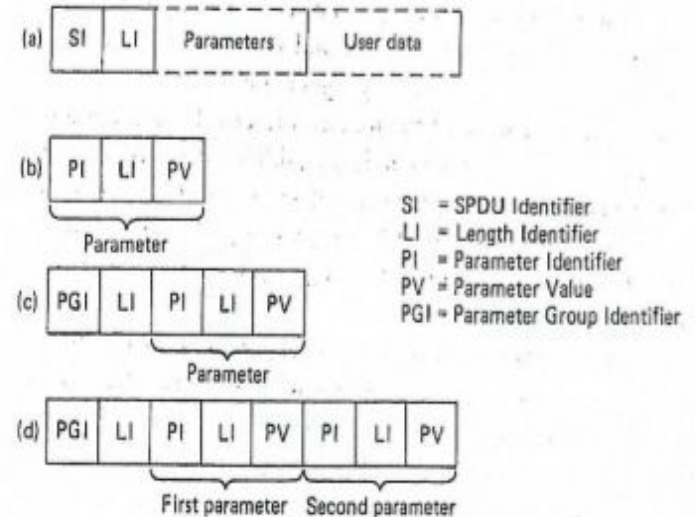

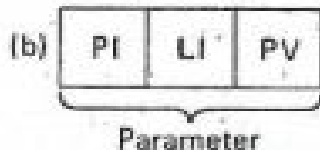


Fig. 7-18. (a) General format of an SPDU. (b) - (d) Examples of the parameters field.

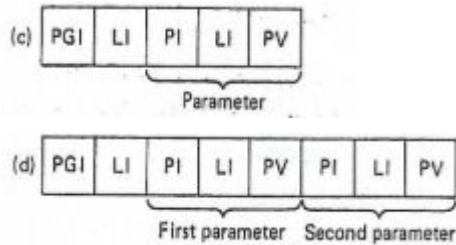


Muitos serviços primitivos da camada de sessão e consequentemente seus SPDUs, carregam parâmetros. Diversos formatos são providenciados para codificar estes parâmetros.

O exemplo mais simples como mostra na figura 7-18(b), possui um campo PI(Parameter Identifier) de 1-byte dizendo qual parâmetro segue, outro campo LI(Length Identifier) de 1-byte indicando quão longo o parâmetro é, e o campo PV(Parameter Value) contendo o valor numérico do parâmetro.



Outros formatos gerais são os exemplos das figuras 7.18(c) e (d).



Nessas SPDUs, parâmetros podem ser coletados em grupos. Cada grupo inicia com um PGI(Parameter Group Identifier) seguido de um campo LI dando o tamanho deste grupo. Os seguintes parâmetros individuais são descritos exatamente como mostrado na figura 7.18(b) . Apesar da figura apenas ilustrar apenas 1 e 2 parâmetros de grupo, grupos maiores também são aceitos. A figura © com grupo de 1 parâmetro é igual que a figura básica (b) . Essas formas de amostragem foram incluídas para compatibilidade com o protocolo teletex CCITT .





# Concatenação e Segmentação


Quando a entidade de sessão constrói uma SPDU, esta pode dar ela a camada de Transporte como uma mensagem. Entretanto, em vários casos, o protocolo de sessão também permite a entidade de sessão empacotar diversos SPDUs juntos em uma única mensagem. Esta técnica chamada concatenação reduz o número de primitivas de transporte que deveriam ser chamadas. O processo inverso (desempacotar uma mensagem chegando em múltiplas SPDUs) feito pela entidade remota de sessão é chamado segmentação. Note que camada de transporte é completamente inconsciente desses dois processos feitos pela camada de sessão. Ambos processos podem ser feitos em outras camadas também.



## Regras de Concatenação

Existem regras bizarras para concatenação. As SPDUs são divididas em 3 categorias: aquelas que podem não ser concatenadas, aquelas que devem ser concatenadas e aquelas que podem ou não, na camada de sessão. Se uma SPDU que deve ser concatenada e tem que ser mandada sem uma outra SPDU para se concatenarem, a entidade de sessão pode gerar um GIVE TOKENS SPDU e adicionar um parâmetro para não pedir nenhum dos tokens, criando uma null SPDU. Outro aspecto peculiar de concatenação é um conjunto adicional de regras especificando a ordem que as SPDUs devem ser grudadas e também ordem que devem ser processadas após serem segmentadas.

O OSI session protocol standard é conceitualmente direto, porém repleto de detalhes. O próprio protocolo não descreve como implementa-lo em detalhe, apenas o descreve como uma larga máquina de estados finitos.



A ARPANET não possui uma camada de sessão ou algo parecido com isso. Possui apenas aplicações para gerir suas sessões. Por outro lado, grande parte do trabalho na RPC ocorreu com a ARPA Internet.

MAP and TOP usam uma forma restrita da camada OSI de sessão. Estabelecimentos de sessões, transferências de dados e lançamento de sessões são totalmente suportados pelo modo Full-Duplex. Os protocolos de sessão MAP/TOP são subsets do protocolo de sessão OSI completo.

Como a ARPANET, USENET não possui uma camada de sessão. Diferente da ARPANET, esta não consegue nem com camadas superiores implementar esses serviços de sessão próprios. Nenhum desses serviços de sessão são necessários.



## 7.4 SUMÁRIO

A camada de sessão é a primeira camada que vimos que não é preocupada com a comunicação pura. Seu trabalho é dar valor a instalação de uma comunicação confiável fornecida pela camada de transporte. Um serviço que é ofertado é o gerenciamento de diálogo, cuidando de quem está a se comunicar em uma conexão half-duplex. Outro serviço é a sincronização, permitindo o roll back e recuperação de erros de não comunicação que foram detectados nas camadas superiores. Diversos outros menores serviços também são oferecidos.