A Camada de Sessão

Lucas Felipe - magmor@ufu.br
Marcus Adriano - marcusadriano@ufu.br
Murilo Santos -

Sumário

7.3. Exemplos da camada de sessão

7.3.1. A Camada de Sessão em Redes Públicas

7.3.2. A camada de sessão na ARPANET

7.3.3. A camada de sessão no ARPANET

7.3.4. A camada de sessão no ARPANET

7.4. Sumário

7.1 - Problema de design da camada de sessão

Nesta seção discute-se sobre o problemas de design da camada de sessão. Que são eles: Gestão de Diálogos, Sincronização, e Gestão de Atividades, entre outros. Todos esses estão acima da Camada de Transporte como serviços de valor agregado.

7.1.1 - Serviços providos à Camada de Apresentação

A Camada de Sessão é responsável por prover serviços para a camada de Apresentação, por meio de Pontos de Acesso ao Serviço de Sessão(SSAP), por comunicar em par com outra Camada de Sessão, em par, por meio de Protocolo de Sessão de Unidades de Dados(SPDU).

A Camada de Sessão utiliza os serviços da Camada de Transporte para realizar suas funções(conexão de usuários, transferência de arquivos, etc), para auxiliar suas entidades e primitivas.

Principais Funções

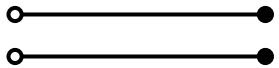
Fornecer uma maneira para que os usuários da sessão estabelecer conexões, chamar sessões e transferir dados sobre elas de maneira ordenada.

Fornecer login remoto de um terminal para outro computador distante, ou para uma transferência de arquivo, ou para qualquer outro propósito através do terminal.

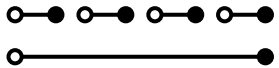
Tipos de Mapear Sessões

○ Conexão estabelecida
● Conexão liberada

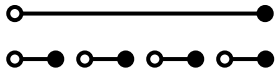
Mapeamento 1-1: quando apenas uma sessão utiliza uma conexão da camada de transporte.
Ex.: Email



Mapeamento N-1: quando apenas várias sessões utilizam uma conexão da camada de transporte.
Ex.: Dois browsers acessam a mesma URL no mesmo computador.



Mapeamento 1-N: quando apenas uma sessão utiliza várias conexões da camada de transporte.
Ex.: Torrent



Tipos de Mapear Sessões

Vale ressaltar que só pode ser feita a ligação de 1-1, ou seja, uma sessão está ligada a uma conexão de transporte, quando a sessão se encerra a conexão de transporte pode continuar por ali, aguardando por outra sessão. Mas o mesmo não vale para a conexão de transporte, que pode em uma só sessão multiplexar várias conexões de transporte, isso pode ser feito para melhorar o desempenho da sessão, essa função é feita pela Camada de Transporte.

7.1.2 - Troca de Dados

A característica marcante da Camada de Sessão é sua troca de dados, e esta possui três fases, estabelecimento, uso e lançamento.

O Estabelecimento da sessão é feito, é feito como na camada de transporte, ou seja, envolve conexão em pares, para definir vários parâmetros(bandeira e qualidade de serviço) da camada de transporte que são apenas repassados, mas também possuem os parâmetros da própria camada. Podendo fazer parecer que a Camada de Sessão apenas repassa informações, mas não é bem assim.

7.1.2 - Troca de Dados

A principal diferença entre uma sessão e uma conexão de transporte, são como elas são lançadas. Em transporte, uma conexão é terminada de forma brusca por meio da primitiva T-DISCONNECT.request, enquanto que na sessão é utilizada a primitiva S-RELEASE.request, onde nenhum dado é perdido.

A figura 7.3 retrata a diferença entre as duas, com o erro que pode acontecer, em conexão de transporte encerra-se somente quando um dos pares decide encerrar, enquanto que na sessão somente se encerra com consenso de ambas as partes.

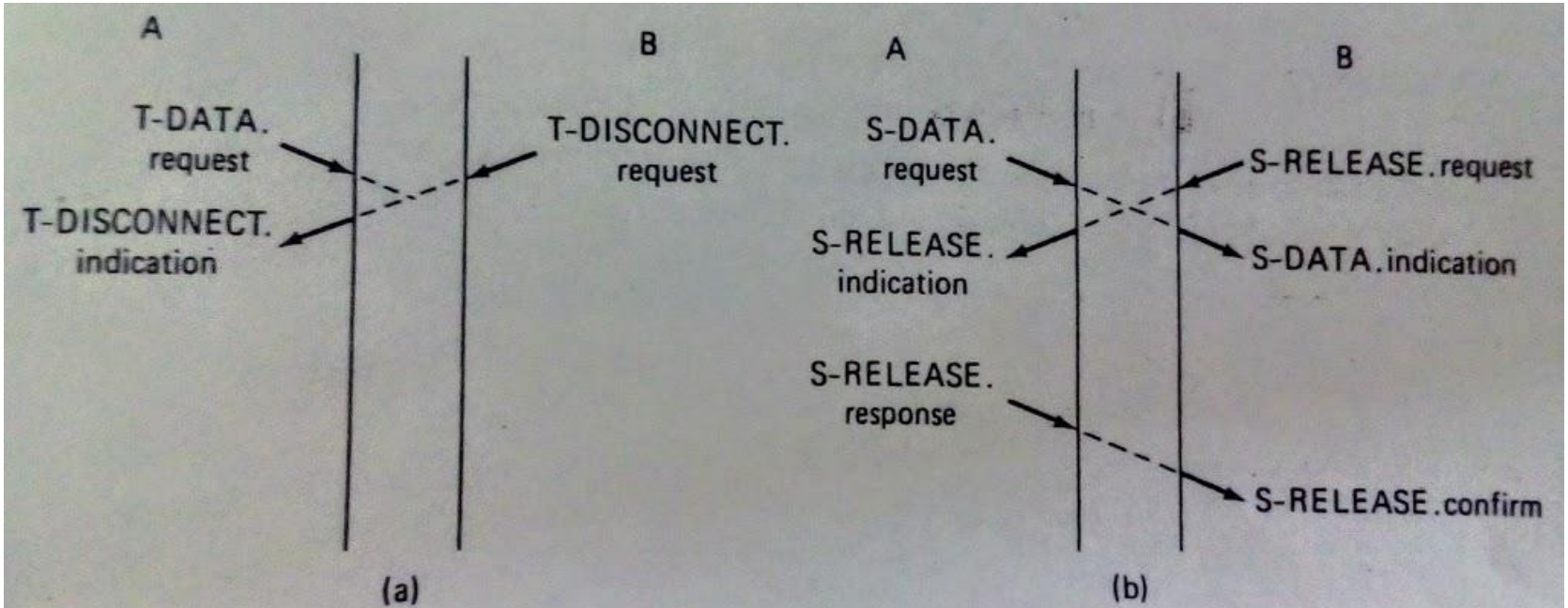


Fig. 7-3. (a) Abrupt release. (b) Orderly release.

7.1.3 - Gerenciamento de Diálogo

O modelo de referência foi criado com a premissa de que todo canal de comunicação seja full-duplex, ou seja, qualquer um dos lados(par) consegue transmitir PDU’s. Porém, existem situações que existe apenas a comunicação half-duplex, ou seja, usuários revezam na conexão. Ex. Banco de Dados.

Para obter essa habilidade, o gerenciamento de diálogo implementa o uso de tokens de dados. No qual quando uma sessão é estabelecida, se selecionada a opção half-duplex, é escolhido o lado que o token irá. Assim somente quem possui o token pode transmitir dados, e o outro par somente recebe dados.

7.1.3 - Gerenciamento de Diálogo

Quando o possuidor do token acaba sua tarefa, ele passa o token para o outro lado, através da primitiva S-TOKEN-GIVE.request, se o outro lado tiver requerido o token. Para isto utiliza-se a primita S-TOKEN-PLEASE.request, assim o usuário do token pode optar por passar o token, ou continuar a utilizá-lo. O usuário só poderá então enviar dados sem o token de duas maneiras, usando dados acelerados providos pela Camada de Transporte, ou quando o canal é full-duplex que desta maneira não irá existir token, e os dois lados podem transmitir dados.

7.1.4 - Sincronização

O serviço de sincronização oferecido pela Camada de Sessão tem como objetivo a recuperação de um estado conhecido de uma entidade caso ocorra erro ou desacordo.

Uma das causas da existência desse serviço nesta camada é devido a Camada de Transporte, mesmo está conseguir consertar erros de comunicação e falhas da sub-rede, porém esta não provide este serviço para as camadas superiores na ocorrência de erros. Ex.: problema na impressora, pausa no download.

Como funciona?

Quando existe uma sessão, o usuário pode escolher quando criar um ponto de sincronização, assim enviando uma primitiva para seu par. Existem dois tipos de pontos, o Menor e o Principal. Quando a sessão invoca a primitiva para a criação de um ponto Principal, este requer uma confirmação do par, diferente do ponto Menor, que não possui confirmação.

A diferença entre os dois, é que quando é requisitado a ressincronização, o último Major é o primeiro ponto e o mais antigo da linha de ressincronização, e pode haver outros pontos menores depois do Principal, mas apenas um ponto principal.

Como funciona?

Definir um ponto de sincronização, maior ou menor, requer a posse dos tokens relevantes. Dois tokens independentes estão disponíveis (para sincronização maior e menor). Eles são distintos um do outro e também distintos do token usado para controlar o fluxo de dados em conexões half-duplex. Quando a ressincronização ocorre, todos os tokens são restaurados para as posições que tiveram no instante em que o ponto de sincronização foi definido.

7.1.5 - Gerenciamento de Atividades

A idéia por trás do gerenciamento de atividades é permitir que o usuário separe o fluxo de mensagens em unidades lógicas chamadas atividades. Cada atividade é completamente independente de quaisquer outras atividades que possam ter passado antes ou depois dela.

Cabe ao usuário marcar o início e fim de cada atividade. Dessa maneira, a sessão conseguirá alcançar esse objetivo e definir para cada transferência de arquivo como uma atividade separada, assim desta maneira o usuário consegue separar o fluxo de mensagens, possibilitando uma melhor coesão(além de prover a sincronização, para caso ocorra erros) de dados para ambos os lados.

Como funciona?

Para iniciar uma atividade o remetente deve emitir a primitiva S-ACTIVITY-START.request, aguardando a resposta do outro lado com um S-ACTIVITY-START.indication, assim é marcado o início do arquivo, o mesmo acontece com o fim do arquivo, com a única diferença é que se usa a primitiva S-ACTIVITY-END (request ou indication).

Para evitar o início de duas atividades ao mesmo tempo, utiliza-se o mesmo token quando o canal é half-duplex, só o usuário de posse do token, pode iniciar uma atividade.

Casos Especiais

Algumas atividades podem se encaixar em um caso especial, como a de um banco quando se realiza uma transação, a atividade não pode ser processada ponto a ponto, à medida que o banco recebe as mensagens, e sim quando a atividade se encerra. Para isso a Camada de Sessão armazena a atividade em um buffer, esse ato é chamado de Quarentena.

Algumas atividades podem ser interrompidas, desta maneira para que as atividades não sejam perdidas e que não precise retornar do início da atividade, existe a primitiva S-ACTIVITY-INTERRUPT.request e para retornar à atividade usa-se a primitiva S-ACTIVITY-RESUME.request.

Problemas e Soluções

Como foi explicado, existe mais de um token, e isto pode vir a ocorrer um deadlock, quando cada par fica com um token e ambos requisitam o outro token através da primita S-TOKEN-PLEASE.request, então ambos ficam loop infinito.

Para contornar essa situação foi proposto um único token onde só iniciar uma atividade quem possui o token, e cada vez que o usuário inicia uma atividade, o serial de sincronização de pontos reinicia em 1 e não existe a possibilidade de re-sincronizar uma atividade anterior após ter começado uma nova.

7.1.6 - Relatório de Exceção

Outro recurso da Camada de Sessão é o Relatório de Exceção, utilizado para relatar os casos inesperados. Quaisquer que seja a exceção o usuário pode emitir a S-U-EXCEPTION-REPORT.request, e pode dentro dessa primitiva enviar até dados da exceção.

Esse serviço serve para inúmeras razões para o usuário, como também para o provedor de serviços que utiliza a primitiva S-U-EXCEPTION-REPORT.indication para reportar ao usuário que sofre com problemas internos nas camadas inferiores ou até mesmo na própria camada.

7.1.7 - As Primitivas de Serviço da Camada de Sessão OSI

Ao todo existem 58 primitivas na Camada de Sessão, e estão subdivididas por serviços que são eles: estabelecer conexão, liberar conexão, troca de dados, gerenciamento de token, sincronização, gerenciamento de atividades e relatório de exceção. E podem ser um dos quatro tipos: requisição, indicação, resposta ou confirmação.

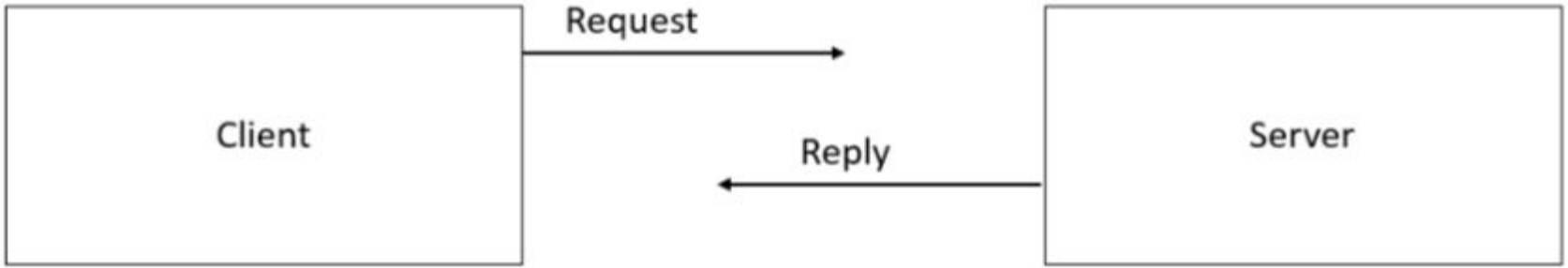
7.2 - Chamada de Procedimento Remoto(RPC)

7.2 - Remote Procedure Call

- A preocupação principal da camada de sessão é gerenciar o diálogo e tratamento de erros ocorridos na camada de transporte;
- Na camada de sessão faz pouco o modelo sem conexão;
- Porém há um trabalho de pesquisa de universidades e indústrias baseado no modelo sem conexão;
- Modelo sem conexão: “host” origem envia quadros independentes ao “host” destino, sem que o “host” de destino confirme o recebimento desses quadros;
- O nome deste modelo é **Remote Procedure Call;**
- Não se encaixa no modelo OSI.

7.2.1 - O Modelo Cliente-Servidor

- Comunicação na forma “request-reply”
- Considerando possível a conexão entre clientes e servidor e a utilização de comunicação half-duplex sobre estas sessões gera um elevado overhead, sendo não atrativo para aplicações como servidor de arquivos, pois a performance é crítica;
- Em redes LAN, o modelo sem conexão é muito bom!


Client
Request
Reply
Server

- Se o problema de performance é resolvido da maneira “sem conexão”, então este modelo possui uma falha: conceitos básicos de comunicação: I/O, interrupção;
- Programas se comunicam com X-DATA.request e X-DATA.indication, de tal forma que isso é um I/O e interrupções;
- Estas dificilmente são as ferramentas para se construir aplicações bem estruturadas.

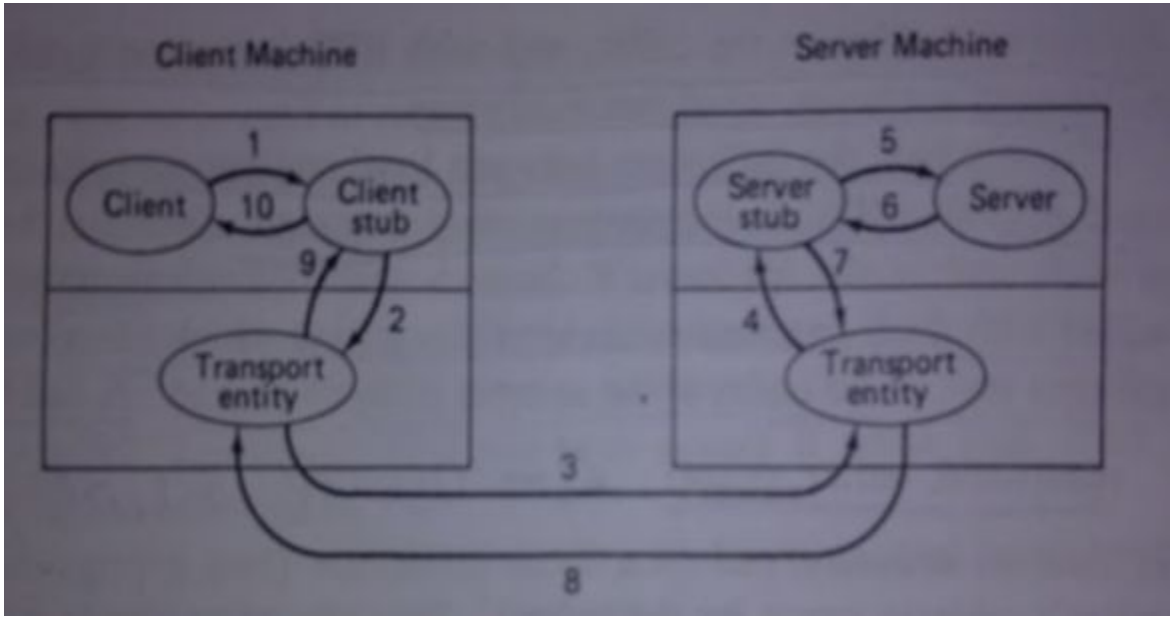
- o RPC aborda o modelo cliente-servidor de uma perspectiva diferente:
  - O cliente envia mensagens para o servidor e recebe respostas
- O modelo RPC funciona parecido como a chamada de um procedimento e recebe um resultado. Em ambos os casos o cliente executa uma chamada de procedimento, aguardo o processamento e recebe um resultado;
- A diferença entre procedimento local e RPC está no fato de o procedimento local executar na própria máquina, enquanto RPC em máquinas diferentes;
- O cliente não consegue distinguir qual procedimento é local ou RPC;

Exemplo: read(fileid, buffer, count)

- O “bonito” desse esquema é o fato de que ocorrem somente chamada de procedimentos, não ocorrendo tratamento direto de I/O e muito menos (o que seria pior) interrupção;
- Procedimentos como o exemplo anterior, do “read”, são chamados de **stubs**;
- No exemplo anterior o stub read transfere dados, mas isso não é obrigatório, qualquer tipo de requisição pode ocorrer;
- Invocando procedimentos **stubs** apropriados, o cliente pode executar ações arbitrárias no servidor.

7.2.2 - Implementação do RPC

- Implementação baseada no trabalho de Birrell e Nelson (1984)
- Implementação com 10 passos, apresentados na próxima figura.



1. Consiste na chamada da procedure stub pelo programa no cliente, o stub coleta os parâmetros e empacota em uma mensagem (**parameter marshalling**)
    a. Passagem de valores;
    b. O cliente age normalmente, pois para ele é uma chamada local.
2. Transmissão ocorre na etapa 2 através da camada de transporte
    a. redes LAN só é colocado um **header** no pacote e ele é colocado na rede;
    b. redes WAN é mais complicado (armadilha para o Sistema Operacional). **[System call]**
3. Da camada de transporte para a camada de Rede;
4. Já no servidor, da camada de transporte para a a **stub do servidor;**
    a. desempacotamento dos parâmetros;

5. Chamada do procedimento da stub no servidor;
6. Produção de retorno pela chamada do procedimento no servidor;
7. Ocorre o **marshalling** (empacotamento) do retorno, passando o pacote para a camada de transporte (assim como o passo 2);
8. O cliente recebe a resposta do servidor;
9. O stub do cliente obtém a resposta;
10. Finalmente o stub do cliente retorna para quem o chamou.

Problemas...

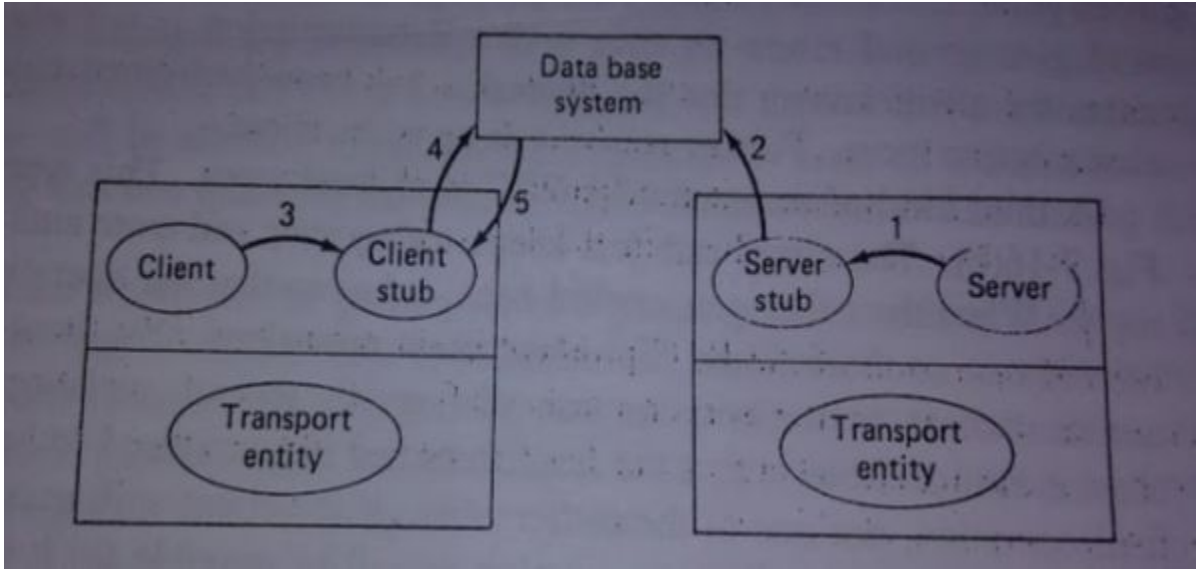
- vale ressaltar que fica transparente para o cliente que é apenas uma chamada de procedimento local;
- Problema da passagem por parâmetro (modelo copiar/restaurar);

Exemplo do Problema da Passagem Por Referência

```
1  program test(output);
2  var a:integer;
3
4  procedure doubleincr(var x, y: integer);
5
6  begin
7  x:= x + 1;
8  y:= y + 1;
9  end;
10
11 begin { main program}
12 a:= 0;
13 doubleincr (a, a);
14 writeln(a);
15 end;
```

- solução: proibir: ponteiros e parâmetros por referência.
- Isso quebra um pouco da transparência.

Ligação entre cliente-servidor



Ligação Cliente-Servidor

- Birrel e Nelson (1984), descreveram um sistema usando banco de dados, contendo o nome do servidor, endereço de rede e identificador único;
- As informações são colocados no BD usando o procedimento ***export*** no servidor;
- Quando o cliente executa um procedimento e a stub precisa localizar o servidor, ela faz uma pesquisa pelo nome do servidor, com isso o BD retorna o identificador e o endereço de rede do servidor procurado;
- Procedimento chamado de **Binding** (ligação);
- Agora o servidor já é conhecido pela stub.

E se o servidor “cair” ou tiver um “crash”??

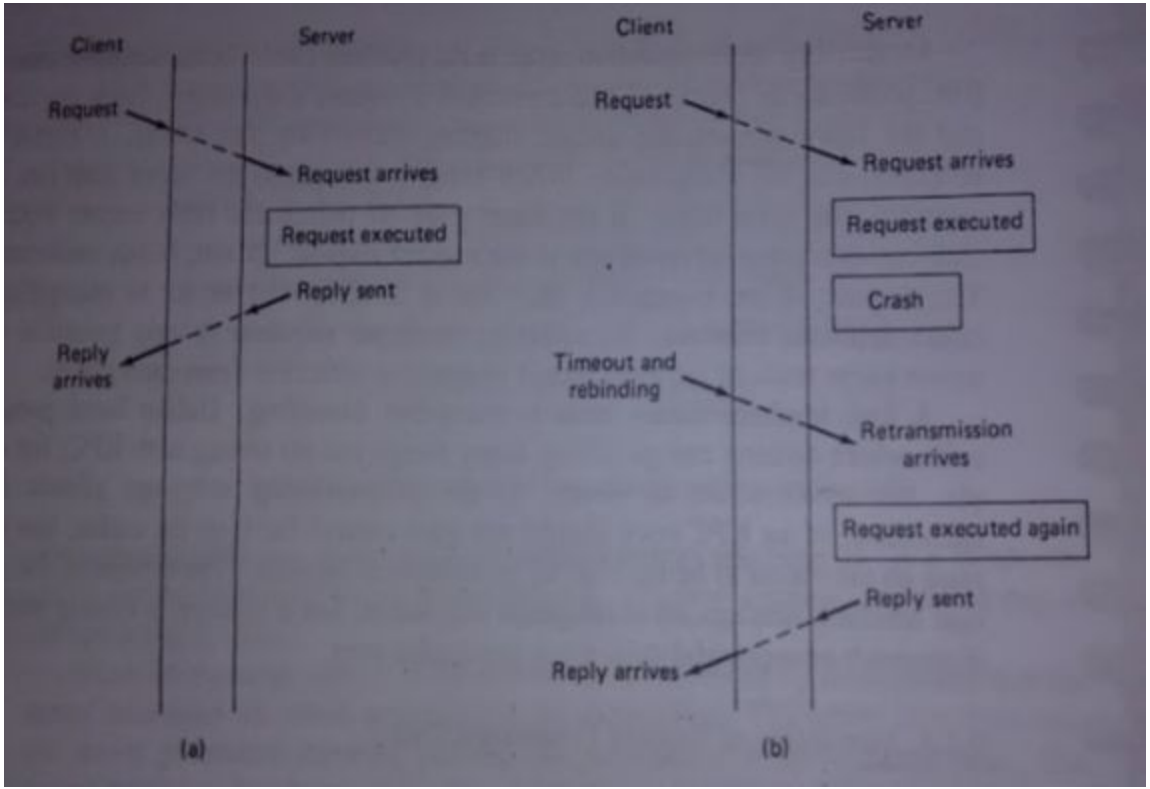
- **rebind** necessário;
- protocolo RPC parecido com Stop-And-Wait;
  - uma mensagem contém identificador do servidor, identificador da transação e os parâmetros;
  - O envio de uma requisição inicializa um timer
  - Quando o timer acabar e se a mensagem de retorno ainda tiver sido recebido, verificar se o servidor recebeu a mensagem e se necessário retransmitir;
  - Salvar os id da transação, ajuda eliminar duplicatas.
- Exception Handler
  - Isso deve ser permitido pela linguagem de programação usada;
  - As mensagens de sucesso/fracasso devem ser distinguidas.

7.2.3 - Semântica do RPC

- Perda de mensagens;
- Servidor “crashou”;
- Cliente “crashou”.

Operação

Normal



- Situação: “servidor cai antes de enviar a resposta”
- Há 3 situações possíveis para o cliente:
  1. Esperar a resposta para sempre
  2. Time out e reportar uma exceção ao cliente
  3. Time out e retransmitir a requisição

1. Esperar a resposta para sempre

- É semelhante quando o cliente chama o procedimento local com loop infinito. Nenhum timer é utilizado e o procedimento nunca terminará.
- Intervenção manual para encerrar o cliente.

2. Time out e reportar uma exceção ao cliente

- reportar ao cliente quando o servidor apresentar alguma falha e logo o time out irá terminar.
- Lançamento de exceções ou reportamento de erro
- Similar quando acontece uma divisão por zero, em procedimentos locais.

3. Time out e retransmissão

- Quando houver alguma problema no servidor e der time out, haverá retransmissão da requisição.
- Se o servidor estiver realizado reregistro na base de dados, depois de um problema qualquer, a retransmissão do cliente sempre será rejeitada na entidade de transporte.
- Após o cliente refazer o **bind** com o servidor, a troca de informações ocorrerá normalmente!

7.2.4 - Órfão

- Impactor de crash de servidor;
- Crash de cliente depois de iniciar uma chamada de procedimento
  - O servidor continua sua execução normalmente
  - O servidor rodando com o cliente em crash, é chamado de Órfão
- Servidores órfãos
  - Uso indevido de recursos
  - lock em arquivos o que pode causar deadlock
  - Após o cliente reiniciar, pode ocorrer um problema de confusão

- Nelson (1981) - Descreveu algumas maneiras de lidar com órfãos:
  - exterminação: após voltar de um crash a máquina verifica se há alguma RPC sendo executado e então é pedido para que o servidor elimine o processo.
    - Para isso é necessário arquivo de log de RPC pendentes:
  - expiração: o servidor precisa de um tempo para completar o procedimento, se ele não terminar em um pequeno intervalo previamente dito, é solicitado ao cliente mais uma fatia de tempo (***quantum***). Se essa solicitação não é atendida, o processo é encerrado.
  - reencarnação: ocorre quando a exterminação não funciona. Particionamento de tempo. Mata todos os processos do servidor.
  - gentil reencarnação: Tenta identificar quem iniciou o processo ao invés de terminá-lo, quando não encontrado, o servidor mata o processo.

7.2.5 - Discussão sobre RPC

1. Interface design
   a. transparência, parâmetro *vs* referência, semântica, “remote” na frente dos procedimentos, tratamento de exceção (dependente de linguagem).
2. Cliente design
   a. time out e órfãos
3. Servidor design
   a. paralelismo
4. Protocolo design
   a. alta performance
   b. primitivas: BIND, UNBIND, INVOKE e RESULT.

7.3 - Exemplos da Camada de Sessão

7.3.1 - A Camada de Sessão em Redes Públicas

PRIMITIVA	REQ	IND	RESP	CONF	SPDU ENVIADA NA REQUISIÇÃO	SPDU ENVIADA NA RESPOSTA
S-CONNECT	X	X	X	X	CONNECT	ACCEPT, REFUSE
S-RELEASE S-U-ABORT S-P-ABROT	X X	X X X	X	X	FINISH ABORT (ABORT)	DISCONNECT, NOT FINISHED (ABORT ACCEPT)
S-DATA S-EXPEDITED-DATA S-TYPED-DATA S-CAPABILITY-DATA	X X X X	X X X X	 X	 X	DATA TRANSFER EXPEDITED DATA TYPED DATA CAPABILITY DATA	 CAPABILITY DATA ACK
S-TOKEN-GIVE S-TOKEN-PLEASE S-CONTROL-GIVE	X X X	X X X			GIVE TOKENS PLEASE TOKENS GIVE TOKENS CONFIRM	 (GIVE TOKENS ACK)

S-ACTIVITY-START S-ACTIVITY-END S-ACTIVITY-DISCARD S-ACTIVITY-INTERRUPT S-ACTIVITY-RESUME	X X X X X	X X X X X	 X X X	 X X X	ACTIVITY START ACTIVITY END ACTIVITY DISCARD ACTIVITY INTERRUPT ACTIVITY RESUME	 ACTIVITY END ACK ACTIVITY DISCARD ACK ACTIVITY INTERRUPT ACK
S-SYNCH-MAJOR S-SYNCH-MINOR S-RESYNCHRONIZE	X X X	X X X	X X X	X X X	MAJOR SYNCH POINT MINOR SYNCH POINT RESYNCHRONIZE	MAJOR SYNCH POINT ACK MINOR SYNCH POINTO ACK RESYNCHRONIZE ACK
S-U-EXCEPTION-REPORT S-P-EXCEPTION-REPORT	X	X X			EXCEPETION REPORT (EXCEPTION REPORT)	

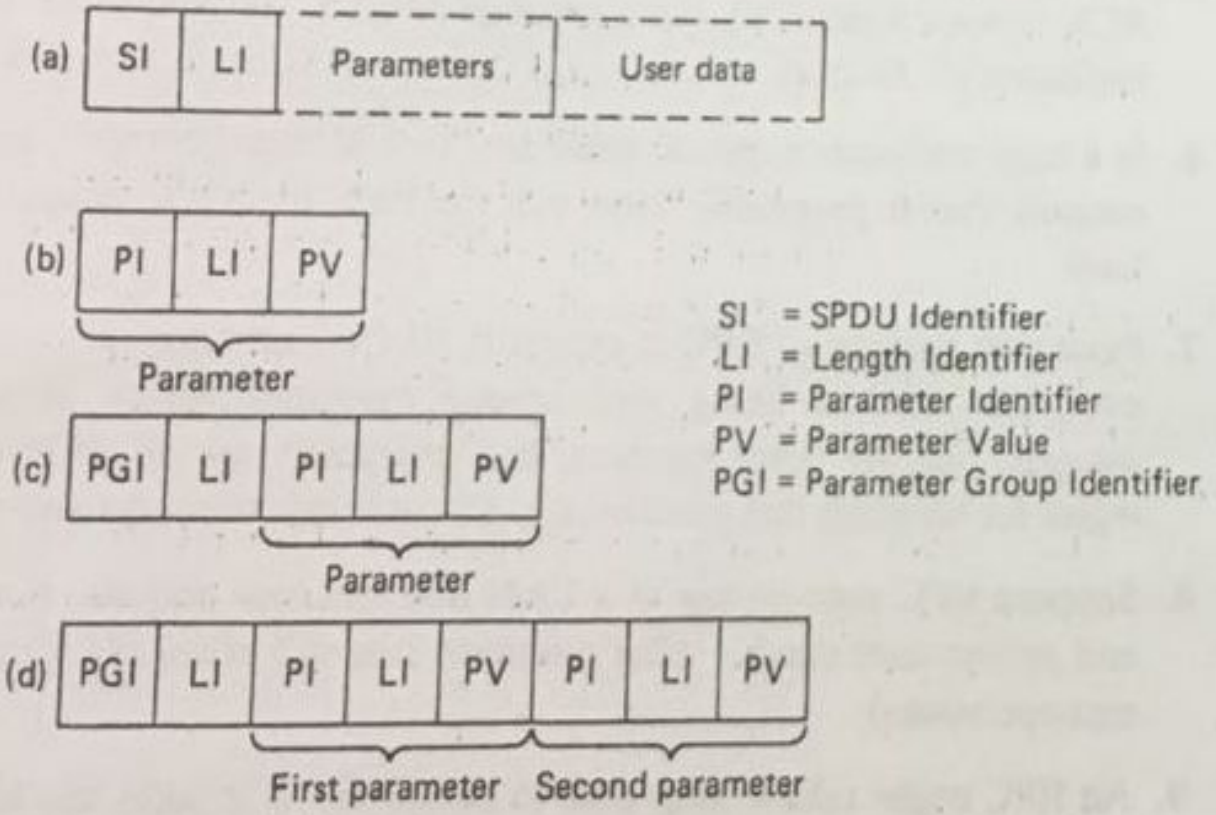


Fig. 7-18. (a) General format of an SPDU. (b) - (d) Examples of the parameters field.

7.3.2 - A Camada de Sessão na ARPANET

- Não possui camada de sessão bem definida.
- As aplicações individuais gerenciam as próprias sessões, se necessário.
- Problemas com a sessão.

7.3.3 - A Camada de Sessão em MAP e TOP

MAP e TOP usam uma forma restrita da camada de sessão OSI. O estabelecimento de sessão, a transferência de dados e o lançamento de sessão são totalmente feitos para o modo full-duplex. O modo half-duplex não é suportado. O serviço de sincronização, gerenciamento de atividades, relatórios de exceção, dados digitados e serviços de capacidade de dados não são necessários.

Os protocolos de sessão MAP / TOP são subconjuntos dos protocolos de sessão OSI completos. Essas SPDU's necessárias para implementar o subconjunto MAP / TOP devem ser implementadas. Os outros são opcionais.

7.3.4 - A Camada de Sessão na USENET

Não possui camada de sessão assim como a ARPANET mas diferente desta, a USENET nem implementa nenhum serviço de sessão, nessa arquitetura a sessão não é necessária.

7.4 - Sumário

É uma camada que não está focada na comunicação bruta mas agrega valor a comunicação confiável provida da camada de transporte.

Oferece serviços de Gerenciamento de diálogo, sincronização, gerenciamento de atividades entre outros.

A camada de sessão é de difícil implementação com vários procedimentos, regras e detalhes. Apesar de ser conceitualmente direta ela é complicada em seus detalhes mas é extremamente útil para melhorar a confiabilidade do sistema.