**6.1.3**

**A camada de serviço de transporte de primitivas osi**

A camanda de serviço de transporte de primitivas osi nos fornece tanto para conexão orientada quanto para sem conexões como podemos ver na figura 6-3

Quando comparamos as figura 6-3 e 5-4 podemos ver que o transporte e o serviço de conexão é bastante similar.

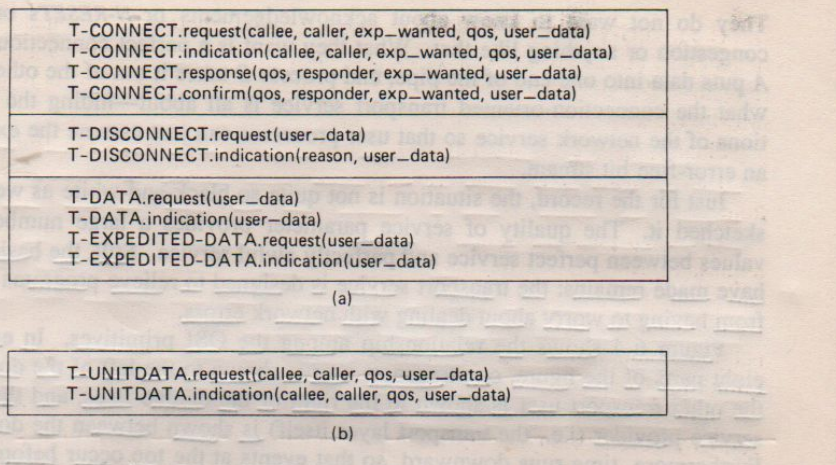
Apesar da similaridade com os serviços de rede também algumas diferenças importantes a principal diferença esta no serviço de rede

Tem como objetivo modelar o serviço oferecido pela rede real.

Essas redes podem perder pacotes e pode emitir N-resets espontaneamente devido a problemas de redes

Sendo assim o serviço de redes nos dão formas de lidar com as confirmações e os N- resets.

Os serviços de transporte ao contrário também não menciona confirmações ou reinícios



Termos utilizados

* Callee – transporte de endereços (tsap) para chamar
* Coller - transporte de endereços (nsap) usando para dar uma flag booleana para verificar se os dados espedidos serão enviados
* Qos – qualidade do serviço designado
* User\_data – 0 ou mais bits transmitidos porem sem ser verificados
* Reason – porque isso aconteceu
* Responder – transporte de endereços conectados ao destinatário

Fig 6-3

a) conexão osi orientada serviço de transporte de primitivas

b) osi sem conexão de transporte de primitivas .

n-reset do ponto de vista de transporte, o serviço esta livre de erros no curso legitimo da rede não esta livre de erros

mas esse é precisamente o propósito da camada de transporte - fornecer serviços confiáveis ​​em cima de uma rede não confiável,

confirmações e N-reset que vêm dos serviços de rede

são interceptados pelas entidades de transporte e os erros são recuperados pelo transporte

protocolo

são interceptados pelas entidades de transporte e os erros são recuperados pelo transporte

protocolo

se uma conexão de rede for reiniciada, o transporte

camada pode apenas estabelecer uma nova e continuar de onde parou com a antiga

outra diferença importante entre o serviço de rede e o serviço de transporte

é para quem os serviços são destinados.

o serviço de rede usado pelo transporte

entidades, que normalmente fazem parte do sistema operacional ou localizado em um

placa ou chip de hardware. poucos usuários escrevem suas próprias entidades de transporte e, portanto, poucos

usuários ou programas sempre veem o serviço de rede simples.

em contraste, muitos usuários não têm uso para as camadas de sessão e apresentação e veem os primitivos de transporte. como mencionamos, o arpanet nem mesmo tem camadas de sessão ou apresentação, então todos os programas que usam a rede

interagir com os primitivos de transporte

(que são diferentes das primitivas de transporte osi, mas são aproximadamente comparáveis).

para ilustrar este ponto, considere os processos

conectado por tubos em unix. eles assumem que a conexão entre eles é perfeita.

eles não querem saber sobre reconhecimentos ou N-RESETs ou congestionamento de rede ou qualquer coisa assim. o que eles querem é uma conexão perfeita. o processo a coloca os dados em uma extremidade do tubo e o processo B os extrai da outra. isso é o que o serviço de transporte orientado à conexão para que os processos possam simplesmente assumir a existência de um fluxo de bits livre de erros.

apenas para registro, a situação não é tão preto e branco como acabamos de esboçar. o parâmetro de qualidade de serviço fornece um grande número de valores cinzentos entre um serviço perfeito e um serviço perfeitamente horrível. ainda assim, o ponto básico que fizemos permanece; o serviço de transporte é projetado para evitar que os programas que o utilizam tenham que se preocupar em lidar com erros de rede.

figura 6-4; mostra o relacionamento entre as primitivas OSI. em cada uma das oito partes da figura, um usuário de transporte é mostrado à esquerda das linhas duplas, o outro usuário de transporte é mostrado à direita das linhas duplas e o provedor de serviços de transporte (ou seja, a própria camada de transporte) é mostrado entre as linhas duplas.

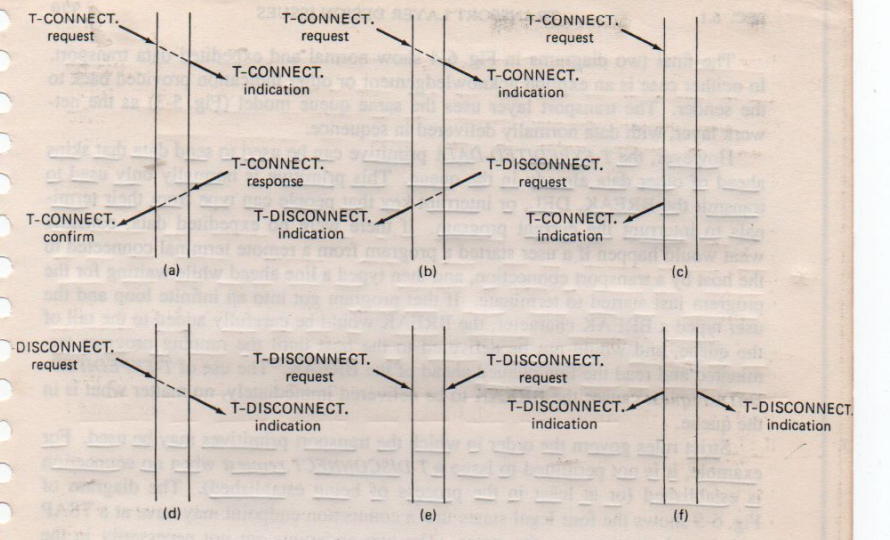
além disso, o tempo corre para baixo, de modo que os eventos no topo são precisos antes dos eventos na parte inferior.

a configuração de conexão normal é ilustrada na fig.6-4. quatro primitivas são usadas. uma das entidades de transporte executa um T-CONEECT. pedido primitivo para sinalizar seu desejo de estabelecer uma conexão com o usuário de transporte anexado ao endereço do ponto de acesso do serviço de transporte (TSAP) nomeado no CONNECT. request primitive.this primitivo resulta em um T-CONNECT.indication com precisão no destino.

o usuário de transporte conectado ao TSAP endereçado obtém a indicação e pode aceitá-la com um T-CONNECT.response, como mostrado na Fig.6-4 (a), ou rejeitá-lo com um T-DISCONNECT. pedido, conforme mostrado na Fig6-4 (b). o resultado de uma aceitação volta para o iniciador como um T-CONNECT. confirme.o resultado de uma rejeição volta como um T-DISCONNECT. indicação

a configuração de conexão normal é ilustrada na fig.6-4. quatro primitivas são usadas. uma das entidades de transporte executa um T-CONEECT. pedido primitivo para sinalizar seu desejo de estabelecer uma conexão com o usuário de transporte anexado ao endereço do ponto de acesso do serviço de transporte (TSAP) nomeado no CONNECT. request primitive.this primitivo resulta em um T-CONNECT.indication com precisão no destino.

o usuário de transporte conectado ao TSAP endereçado obtém a indicação e pode aceitá-la com um T-CONNECT.response, como mostrado na Fig.6-4 (a), ou rejeitá-lo com um T-DISCONNECT. pedido, conforme mostrado na Fig6-4 (b). o resultado de uma aceitação volta para o iniciador como um T-CONNECT. confirme.o resultado de uma rejeição volta como um T-DISCONNECT. indicação



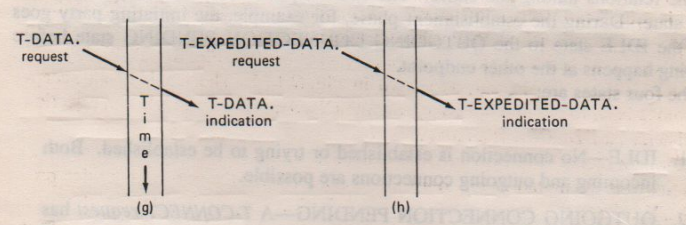


Fig.6-4 algumas sequências válidas de primitivas de transporte osi.

1. Configuração de conexão
2. conexão rejeitada pelo usuario chamado
3. conexão rejeitada pela camada de transporte
4. conexão realizada normalmente
5. liberação simultânea por ambos os lados
6. camada de transporte iniciada a liberaçao
7. transferência de dado normal
8. transferência de de dado expedido

a conexão foi encerada. claramente, um provedor de serviços de transporte bem projetado

não deve emitir primitivas de indicação T-disconect espontâneas muito levianamente, mas

existe circunstância em nenhuma outra ação é possível. Por exemplo

se a rede subjacente falhar e se recusar a reagir a repetidas

tenta se comunicar com ele, há pouco mais o provedor de serviços de transporte

pode fazer do que

quebrar todas as conexões. a menos que a camada de sessão tenha tomado precauções espaciais

contra este problema, a falha terá que ser reportada ao nível mais alto,

e pode exigir intervenção humana para repetir o comando com falha.

O diagrama da figura 6-4 o normal e transporte de dados transmitidos

em nenhum dos casos um reconhecimento explícito ou indicação externa é devolvido ao remetente. o transporte usa o mesmo modelo de fila (fig. 5-3) que a camada de rede, com os dados normalmente entregues em sequência.

o programa começou a terminar. se esse programa entrasse em um loop infinito e o uso digitasse um caractere de quebra, a violação seria cuidadosamente adicionada à cauda de

a fila e não seria entregue ao host até que o programa em execução terminasse e lesse a linha antes do intervalo. o uso de T-expedited-data.request faz com que a quebra seja entregue imediatamente, não importa o que esteja na fila

regras estritas governam a ordem em que as primitivas de transporte podem ser usadas.

Por exemplo, não é permitido emitir uma desconexão T. solicitação

quando nenhuma conexão é estabelecida

(ou pelo menos processo de início estabelecido).

o programa de

fig 6-5 mostra os quatro estados legais que os pontos de extremidade de uma conexão podem ter em um stap

e as relações entre os estados. os dois endpoits não estão necessariamente no

mesmo estado. durante as fases de estabelecimento, por exemplo, a parte inicial vai do estado inativo para o estado de peding de conexão de saída antes

qualquer coisa acontece no outro ponto de extremidade.

os quatro estados são

conexão inativa é estabelecida ou tentar estabelecer conexões de entrada e saída são possíveis. conexão de saída pendente de uma conexão t. pedido foi feito. a forma de resposta do peer remoto ainda não foi recebido conexão de entrada pendente- a t-connect.indication entrou. ele ainda não foi aceito ou rejeitado.

conexão estabelecida - conexão válida foi estabelecida. a fase de estabelecimento concluída e a transferência de dados pode

Parte superior do formulário

pagina 391

6.2.2 establishing e connection

Estabelecer uma conexão parece fácil, mas na verdade é surpreendentemente complicado, especialmente em uma rede do tipo C. À primeira vista, pareceria suficiente para uma entidade de transporte apenas enviar um CR (PEDIDO DE CONEXÃO) TPDU para o destino e aguardar uma resposta CC (CONFIRMAR DE CONEXÃO). o problema ocorre quando a rede pode perder, pode perder, armazenar e duplicar pacotes.

imagine uma sub-rede tão congestionada que as confirmações nunca voltam no tempo, e cada pacote atinge o tempo limite e é retransmitido duas ou três vezes. suponha que a sub-rede use datagramas na rede e cada pacote siga uma rota diferente. alguns dos pacotes podem ficar presos em congestionamentos e demorar para chegar, ou seja, ficam armazenados na sub-rede e saem muito depois.

O pior pesadelo possível é o seguinte: um usuário estabelece uma conexão com um banco, envia mensagens dizendo ao banco para transferir uma grande quantia de dinheiro para a conta de uma pessoa não totalmente confiável e, em seguida, cancela a conexão.

infelizmente, cada pacote lançado, todos os pacotes saem da sub-rede e

chegue ao destino em ordem, pedindo ao banco para estabelecer uma nova conexão, transferir dinheiro (novamente) e liberar a conexão. o banco não tem como saber que são duplicatas, presume que se trata de uma segunda transação independente e transfere o dinheiro novamente. no restante desta seção, estudaremos o problema das duplicatas atrasadas, com ênfase especial nos algoritmos para estabelecer a conexão de uma maneira realizável, de modo que pesadelos como o anterior não possam acontecer.

o ponto crucial do problema é a existência de duplicatas atrasadas. pode ser feito de várias maneiras, nenhuma delas muito satisfatória. uma maneira é usar endereços TSAP descartáveis. nessa abordagem, cada vez que um endereço TSAP é necessário, um endereço novo e exclusivo é gerado, normalmente com base na hora atual.

Quando uma conexão é liberada são descartados para sempre, esta estratégia torna o processo servir da FIG.6-11 impossível.

Outra possibilidade é dar a cada conexão um identificador de conexão (ou seja, um número de sequência incrementado para cada conexão estabelecida), escolhido pela parte inicial e colocado em cada TPDU, incluindo aquele que está solicitando a conexão.

depois que cada conexão é liberada, cada entidade de transporte pode atualizar uma tabela listando conexões obsoletas como pares (entidade de transporte de par, identificador de conexão). sempre que uma solicitação de conexão chegasse, ela poderia ser verificada na tabela para ver se pertencia a uma conexão liberada anteriormente.

infelizmente, esse esquema tem uma falha básica: exige que cada entidade de transporte mantenha uma certa quantidade de informações de histórico indefinidamente. se uma máquina travar e perder sua memória, ela não saberá mais quais identificadores de conexão já foram usados.

em vez disso, precisamos seguir um rumo diferente. em vez de permitir que pacotes vivam para sempre dentro da sub-rede, devemos conceber um mecanismo para matar compactadores muito antigos que ainda estão vagando por aí. se pudermos garantir que nenhum pacote dura mais do que algum tempo conhecido, o problema se torna um pouco mais gerenciável.

o tempo de vida do pacote pode ser restrito a um máximo conhecido usando uma das seguintes técnicas;

1. design de sub-rede restrita

2. colocar um contador de saltos em cada pacote

3. carimbo de tempo de cada pacote

o primeiro método inclui qualquer método que evite que os pacotes entram em loop, combinado com alguma maneira de limitar o atraso de congestionamento no (agora conhecido) caminho mais longo possível. o segundo método consiste em ter a contagem de saltos incrementada cada vez que o pacote é encaminhado. o protocolo de enlace de dados simplesmente descarta qualquer pacote anterior ao tempo acordado. este último método requer o IMP

relógios a serem sincronizados, o que em si é uma tarefa nontrivial, a sincronização é obtida externamente à rede, por exemplo, ouvindo WWV ou alguma outra estação de rádio que transmite a hora exata periodicamente.

na prática, precisaremos garantir não apenas que um pacote está morto, mas também que todas as confirmações a ele também estão mortas, portanto, agora introduziremos T, que é um pequeno múltiplo do verdadeiro tempo de vida máximo do pacote. o múltiplo é dependente do protocolo e simplesmente tem o efeito de tornar T mais longo. se assistirmos um tempo T após o envio de um pacote, podemos ter certeza de que todos os traços dele desapareceram e que nem ele nem seus reconhecimentos apareceram repentinamente do nada para complicar as coisas.

com o tempo restante do pacote limitado, é possível conceber uma maneira infalível de estabelecer conexões com segurança. o método descrito abaixo é devido a tomlinson (1975).

ele resolve o problema, mas introduz algumas peculiaridades próprias. Mas o método foi mais refinado por Sunshine e Dalal (1978).

para contornar o problema de a máquina perder toda a memória ou onde ela estava após um travamento, tomlinson propôs equipar cada host com um relógio de hora do dia. os relógios em hosts diferentes não precisam ser sincronizados. cada relógio assume a forma de um contador binário que se incrementa em intervalos uniformes. além disso, o número de bits nos números de sequência. por último e mais importante, presume-se que o relógio continuará funcionando mesmo se o host for desativado.

a ideia básica é garantir que duas TPDUs com numeração idêntica nunca fiquem pendentes ao mesmo tempo. quando uma conexão é configurada, os k bits de ordem inferior do relógio são usados ​​como o número de sequência inicial (também k bits). portanto, ao contrário de nossos protocolos do capítulo 4, cada conexão começa a numerar suas TPDUs com um número de sequência diferente. o espaço de sequência deve ser tão grande (por exemplo, 32 bits) que no longo prazo. esta relação linear entre o tempo e os números de sequência é mostrada na fig. 6-12

a ideia básica é garantir que duas TPDUs com numeração idêntica nunca fiquem pendentes ao mesmo tempo. quando uma conexão é configurada, os k bits de ordem inferior do relógio são usados ​​como o número de sequência inicial (também k bits). portanto, ao contrário de nossos protocolos do capítulo 4, cada conexão começa a numerar suas TPDUs com um número de sequência diferente. o espaço de sequência deve ser tão grande (por exemplo, 32 bits) que, no final do tempo. esta relação linear entre o tempo e os números de sequência é mostrada na fig. 6-12, o número de sequência inicial para uma conexão aberta no tempo x será x .imagine que em t = 30 seg, um

|  |  |
| --- | --- |
|  |  |

a ideia básica é garantir que duas TPDUs com numeração idêntica nunca fiquem pendentes ao mesmo tempo. quando uma conexão é configurada, os k bits de ordem inferior do relógio são usados ​​como o número de sequência inicial (também k bits). portanto, ao contrário de nossos protocolos do capítulo 4, cada conexão começa a numerar suas TPDUs com um número de sequência diferente. o espaço de sequência deve ser tão grande (por exemplo, 32 bits) que, no final do tempo. esta relação linear entre o tempo e os números de sequência é mostrada na fig. 6-12, o número de sequência inicial para uma conexão aberta no tempo x será x .imagine que em t = 30 seg, um

a) Os TPDUs não podem entrar na região proibida.

b) o problema de ressincronização

A TPDU de dados comuns enviada em uma conexão 5 (aberta anteriormente) recebe o número de sequência 80. chame esta TPDU X. imediatamente após o envio do TPDU X, o host trava e rapidamente

reiniciar. em t = 60, ele começa a reabrir conexões o

Por meio de 4. em t = 70, ele reabre as conexões 5 usando

número de sequência 70, conforme necessário. dentro dos próximos 15 segundos, ele envia dados TPDUs 70 a 80. portanto, em t = 85

uma nova TPDU com número de sequência 80 e conexão 5 foi injetada na sub-rede, infelizmente,

TPDU X ainda existe. se chegar ao receptor antes do novo TPDU 80, será aceito e o TPDU correto rejeitado.

cenário de três protocolos para estabelecer uma conexão um handsharke de três vias (a) operação normal.

B) CR duplicado antigo aparecendo do nada.

c) duplicar cr e duplicar ack.

TPDUs de controle duplicado na figura 6-13 (b) o primeiro TPDU é uma duplicata atrasada

CR

de uma conexão desde

liberado. este TPDU chega a B sem o conhecimento de A.

B reage a este TPDU enviando A um TPDU CC,

na verdade, pedindo a verificação de que

a estava realmente tentando estabelecer uma nova conexão.

quando um rejeita a tentativa de B de

estabelecer, B percebe que foi enganado por uma duplicação atrasada e abandona a conexão

o pior caso quando um CR atrasado e uma confirmação

um cc são

flutuando na sub-rede. este caso é mostrado na figura 6-13 (c) como no exemplo anterior, B obtém um CR atrasado e responde a ele neste ponto, é crucial perceber que B propôs usar y como número de sequência inicial para o tráfego de B para A ,

sabendo muito bem que nenhum TPDU contendo o número de sequência y ou reconhecimentos para y ainda existe. quando o segundo TPDU atrasado chega em B, o

o fato de que z foi reconhecido em vez de y diz a B que esta também é uma duplicata antiga

***pag 380, 394, 396 por foto***