Sistemas Distribuídos

Aula 09 – Tolerância a Falha

DCC/IM/UFRRJ
Marcel William Rocha da Silva

Objetivos da aula

- **Aula anterior**
  - Consistência e replicação
- **Aula de hoje**
  - Modelos de falha
  - Resiliência de processo
  - Comunicação confiável cliente-servidor
  - Comunicação confiável em grupo

Conteúdo Programático

- Introdução e visão geral
- Princípios de sistemas distribuídos
  - Arquiteturas
  - Processos
  - Comunicação
  - Nomeação
  - Sincronização
  - Consistência e replicação
  - Tolerância à falha
  - Segurança
- Seminários

Tolerância a falha

- Diferente dos sistemas convencionais, em SDs as falhas parciais não param totalmente o sistema
  - **Falhas parciais** → problemas isolados em componentes de um SD

- SD deve ser capaz de se recuperar automaticamente de falhas parciais sem prejudicar seu funcionamento

Conceitos básicos

- Existe uma forte relação entre **tolerância a falhas** e **confiança**

- Confiança abrange uma série de requisitos úteis para sistemas distribuídos (Kopetz e Veríssimo, 1993):
  - Disponibilidade
  - Confiabilidade
  - Segurança
  - Capacidade de Manutenção

Conceitos básicos

- **Disponibilidade**
  - Propriedade de um sistema estar pronto para ser usado imediatamente
  - Probabilidade do sistema estar funcionando corretamente em qualquer momento específico e estar disponível para executar suas funções em nome de seus usuários
  - Alta disponibilidade: mais provável de estar funcionando em dado instante de tempo

Conceitos básicos

- **Confiabilidade**
  - Propriedade de um sistema poder funcionar continuamente sem falha
  - É definida em termos de um intervalo de tempo em vez de um instante de tempo
  - Alta confiabilidade: mais provável de continuar a funcionar sem interrupção durante um período de tempo relativamente longo

Conceitos básicos

- **Segurança**
  - Garantia de que se um sistema deixar de funcionar corretamente durante um certo tempo, nada de catastrófico acontecerá
  - Ex.: Sistemas de controle de processos usados em usinas de energia nuclear

Conceitos básicos

- **Capacidade de manutenção**
  - Facilidade com que um sistema que falhou possa ser consertado
  - Sistemas de alta capacidade de manutenção também pode mostrar alto grau de disponibilidade, em especial se as falhas puderem ser detectadas e reparadas automaticamente

Conceitos básicos

- Tolerância a falhas
  - Construção de sistemas confiáveis está intimamente relacionada com controle de falhas
  - O nosso objetivo é estudar casos onde um sistema pode prover serviços, mesmo na presença de falhas

Tipos de falha

- **Transiente**
  - Ocorre uma vez e depois desaparece
  - Se a operação for repetida, a falha não acontecerá novamente
- **Intermitente**
  - Ocorre e desaparece por “sua própria vontade”
  - Difícil de diagnosticar
  - Ex.: conector com problemas
- **Permanente**
  - Continua a existir até que o componente faltoso seja substituído
  - Ex: bugs de software, chips queimados

Modelos de falhas

- Cristian (1991) e Hadzilacos e Toueg (1993)

Tipo de falha	Descrição
Falha por queda	O servidor pára de funcionar, mas estava funcionando corretamente até parar.
Falha por omissão *Omissão de recebimento* *Omissão de envio*	O servidor não consegue responder a requisições que chegam O servidor não consegue receber mensagens que chegam O servidor não consegue enviar mensagens
Falha de temporização	A resposta do servidor se encontra fora do intervalo de tempo
Falha de resposta *Falha de valor* *Falha de transição de estado*	A resposta do servidor está incorreta O valor da resposta está errado O servidor se desvia do fluxo de controle correto
Falha arbitrária	Um servidor pode produzir respostas arbitrárias em momentos arbitrários

Tabela 8.1 Diferentes tipos de falhas.

Usando replicação para mascarar falhas

- **Redundância de informação**
  - Bits extras são adicionados para permitir recuperação de bits deteriorados
  - Ex: FEC (*Forward Error Correction*)
- **Redundância de tempo**
  - Uma ação é realizada e, então, se for preciso, ela é executada novamente
  - Ex: Transações podem ser repetidas, caso tenham sido abortadas
- **Redundância física**
  - Componentes físicos (ou *software*) replicados
  - Ex: RAID (*Redundant Array of Inexpensive Disks*)

Estratégias de tolerância a falha

- Falhas de processos
  - **Resiliência de processos**: replicação de processos em grupos
- Falhas de comunicação
  - **Comunicação confiável cliente-servidor**: Canal de comunicação pode exibir falhas por queda, por omissão, arbitrárias
  - **Comunicação confiável de grupo**: Como implementar entrega confiável de mensagens a todos os processos?
  - **Comprometimento distribuído**: Envolve a realização de uma operação por cada membro de um grupo de processos ou por absolutamente nenhum

Resiliência de processo

- **Ideia básica** → Organizar vários processos idênticos em um grupo

- Como conseguir o objetivo?
  - Questões de projeto:
    - **Grupos Simples** versus **Grupos Hierárquicos**
    - **Associação** a um grupo
  - Mascaramento de falhas e replicação
  - Acordo em sistemas com falha
  - Detecção de falha

Resiliência de processo

- **Premissas**:
  - Processos idênticos são organizados em **grupos**
    - Quando uma mensagem é enviada a um grupo, **TODOS** os membros a recebem
    - Se um processo do grupo falhar, espera-se que um outro se encarregue do tratamento da mensagem
  - Grupos de processos são **dinâmicos**
    - Grupos podem ser criados e removidos
    - Processos podem se mover entre os grupos

Resiliência de processo: Grupos

- **Grupo Simples**
  - Todos os processos são iguais
  - Decisões são tomadas coletivamente
  - Vantagens
    - Não tem ponto de falha único → mesmo que um processo “caia”, o grupo continua a oferecer o serviço
  - Desvantagem
    - Tomada de decisão pode ser complicada, com necessidade de uma votação → retardo

Resiliência de processo: Grupos

- **Grupo Hierárquico**
  - Existe um processo **coordenador**, os demais são denominados "**operários**"
  - Sempre que uma requisição é gerada, é enviada ao coordenador
  - O coordenador decide qual é o operário mais adequado para executá-la
  - Vantagens
    - Decisões são centralizadas
  - Desvantagem
    - Caso o coordenador falhe, o serviço falhará

Resiliência de processo: Grupos

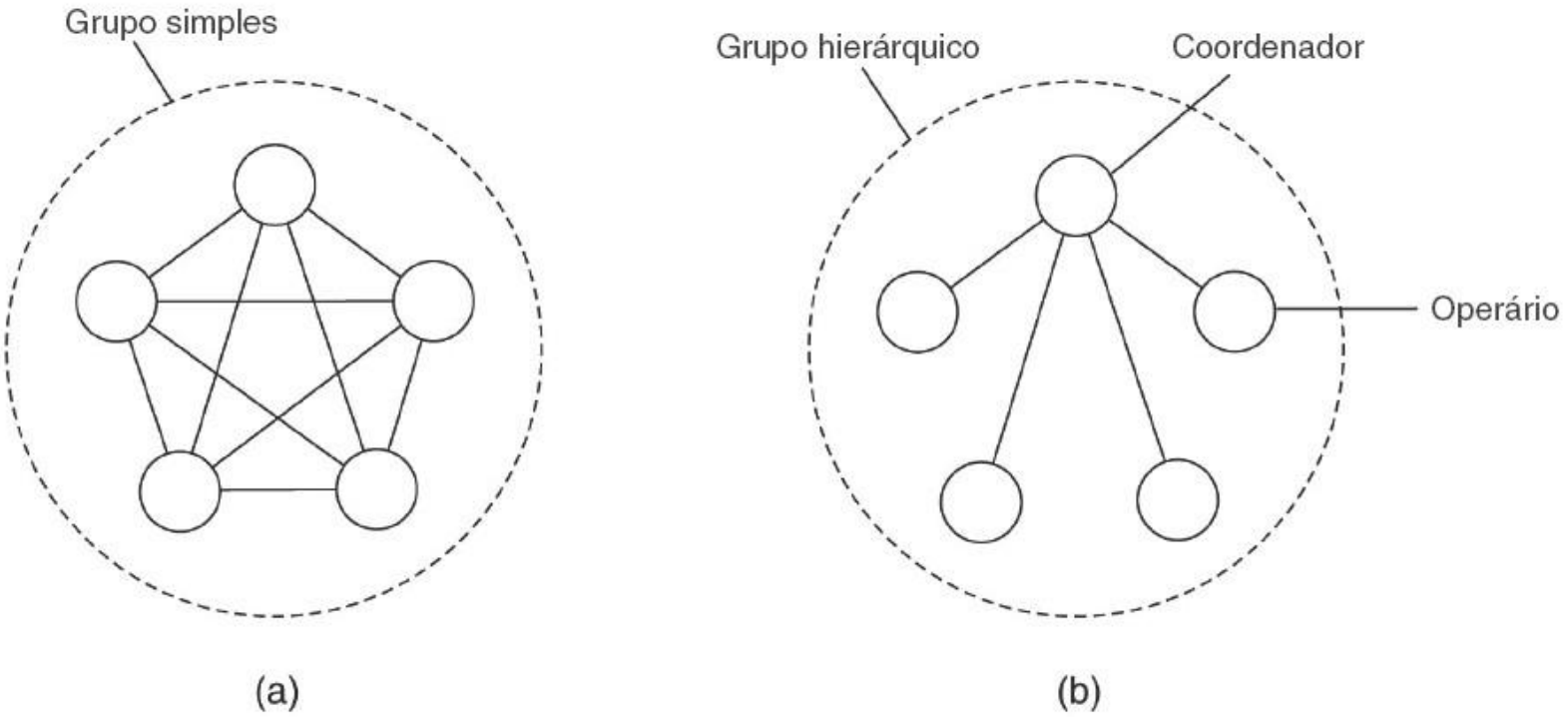


Figura 8.2 (a) Comunicação em um grupo simples. (b) Comunicação em um grupo hierárquico simples.

Resiliência de processo: Associação

- Como criar e eliminar grupos?
- Como permitir que processos se juntem e saiam dos grupos?

- Duas possibilidades:
  1. Servidor de grupos
  2. Gerenciamento distribuído

Resiliência de processo: Associação

- **Servidor de Grupos**
  - Todas as requisições são enviadas a este servidor
  - Mantêm um banco de dados completo de todos os grupos e seus associados
  - Método direto, eficiente
  - Desvantagem
    - Se o servidor cair, o gerenciamento deixa de existir
    - Grupos deverão ser reconstituídos do zero

Resiliência de processo: Associação

- **Gerenciamento Distribuído**
  - Se existe multicast confiável, um processo pode enviar uma mensagem a todos os membros do grupo anunciando que deseja se juntar ao grupo
  - Para sair de um grupo, o processo deveria mandar uma mensagem
  - No entanto, difícil de prever saídas → geralmente causadas por quedas!
  - Entrar/Sair de um grupo devem ser síncronos com as mensagens enviadas/recebidas

Mascaramento de falhas e replicação

- **Terminologia**: Quando um grupo de processos deseja mascarar **k** falhas simultâneas, é classificado como um grupo **k-tolerante a falha** (*k-fault tolerant*)

- **Problema**: Quão grande **k** precisa ser?
  1. Falhas "silenciosas" → Se **k** processos pararem sem propagar informações erradas, basta ter **k+1** processos
  2. Se os processos exibirem falhas e continuarem a enviar respostas erradas as requisições, é preciso um mínimo de **2k+1** processadores para conseguir k-tolerância

Mascaramento de falhas e replicação

- Na prática, é difícil imaginar circunstâncias em que poderíamos dizer com certeza que **k** processos podem falhar embora **k+1** processos não possam falhar!

Acordo em sistemas com falha

- **Objetivo** → Todos os processos que não apresentam falhas devem chegar a um consenso sobre alguma questão, dentro de um número finito de etapas
  - Ex: eleger um coordenador, decidir a validação de uma transação, repartir tarefas entre operários, sincronização, etc

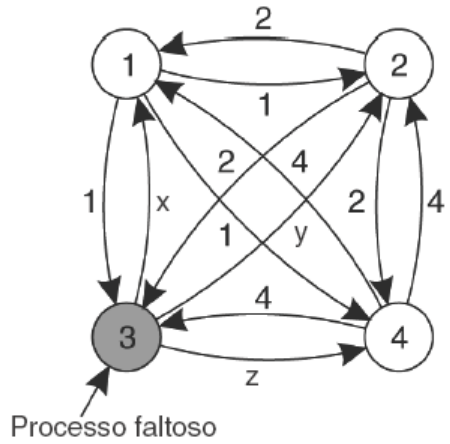
- Problema: Casos em que os processos não podem ser considerados perfeitos

Acordo em sistemas com falha

- Lamport *et al.* (1982)

- **Premissa:** Processos síncronos, mensagens unicast, ordenação preservada e o atraso de comunicação limitado
- **Sistema**: N processos e cada processo ***i*** fornece um valor v_i aos demais
- **Objetivo**: Cada processo deve construir um vetor ***V*** de comprimento ***N*** tal que, se o processo ***i*** não for faltoso, $V[i] = v_i$. Caso contrário, ***V[i]*** é indefinido

Acordo em sistemas com falha



(a)

1	Obteve (1, 2, *x*, 4)
2	Obteve (1, 2, *y*, 4)
3	Obteve (1, 2, 3, 4)
4	Obteve (1, 2, *z*, 4)

(b)

1 Obteve	2 Obteve	4 Obteve
(1, 2, *y*, 4)	(1, 2, *x*, 4)	(1, 2, *x*, 4)
(*a*, *b*, *c*, *d*)	(*e*, *f*, *g*, *h*)	(1, 2, *y*, 4)
(1, 2, *z*, 4)	(1, 2, *z*, 4)	(*i*, *j*, *k*, *l*)

(c)

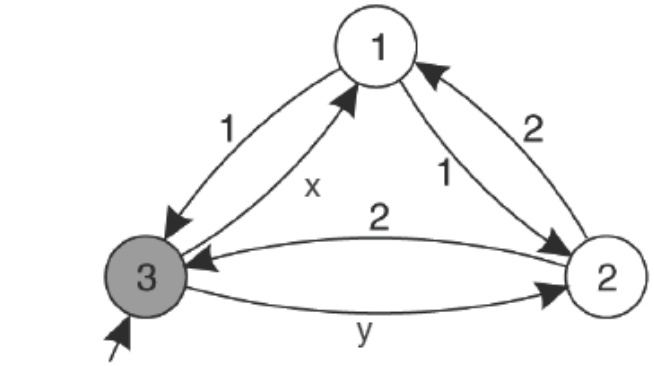
Figura 8.4 Problema do acordo bizantino para três processos não faltosos e um faltoso. (a) Cada processo envia seu valor aos outros. (b) Vetores que cada processo monta com base em (a). (c) Vetores que cada processo recebe na etapa 3.

Acordo em sistemas com falha

- Lamport *et al.* (1982)

"Em um sistema com **k** processos faltosos, pode-se conseguir um acordo somente se estiverem presentes **2k+1** processos funcionando corretamente, para um total de **3k+1**. Um acordo somente é possível se **mais do que dois terços** dos processos estiverem funcionando adequadamente."

Acordo em sistemas com falha



(a)

1 Obteve (1, 2, *x*)
2 Obteve (1, 2, *y*)
3 Obteve (1, 2, 3)

(b)

1 Obteve	2 Obteve
(1, 2, *y*)	(1, 2, *x*)
(*a*, *b*, *c*)	(*d*, *e*, *f*)

(c)

Figura 8.5 *Igual à Figura 8.4, exceto que, agora, há dois processos corretos e um processo faltoso.*

Acordo em sistemas com falha

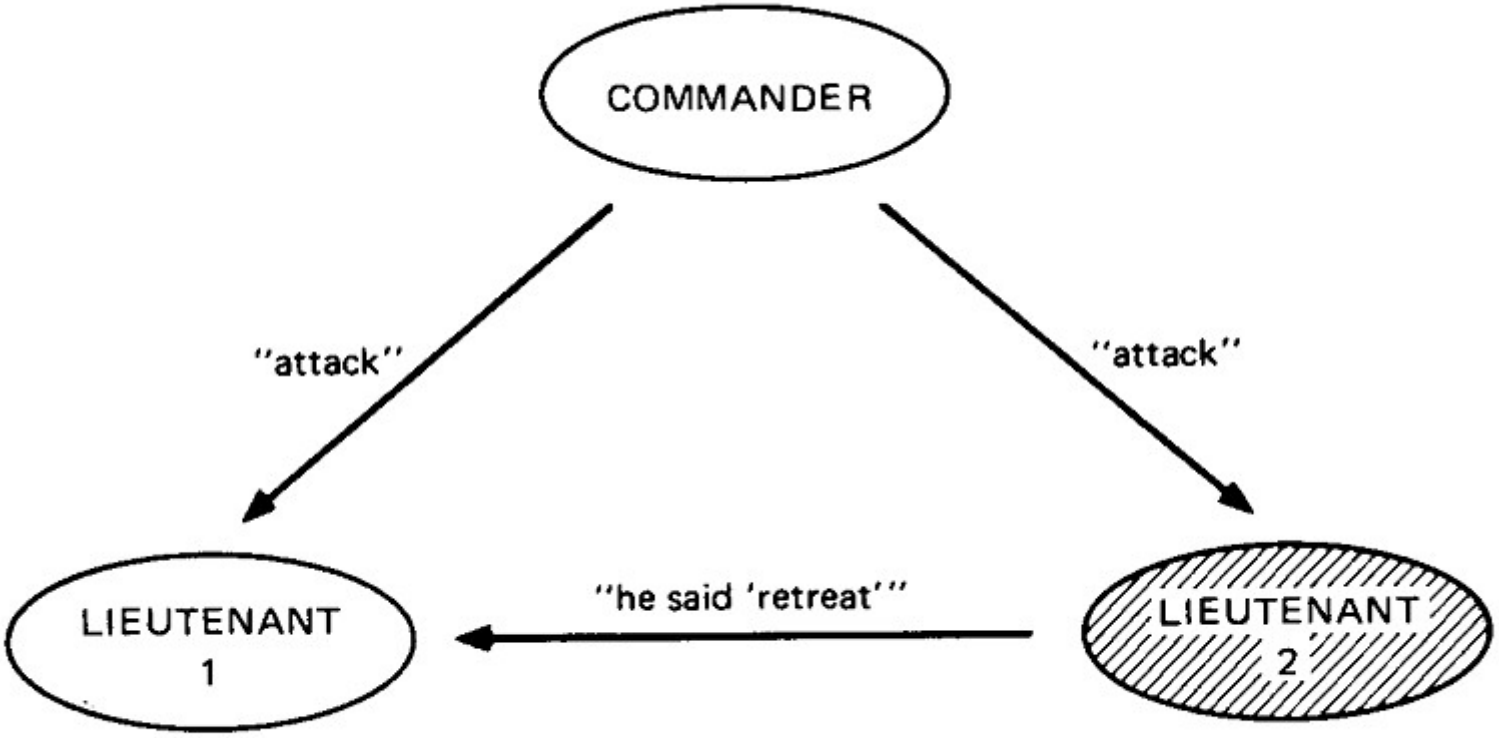


Fig. 1. Lieutenant 2 a traitor.

Acordo em sistemas com falha

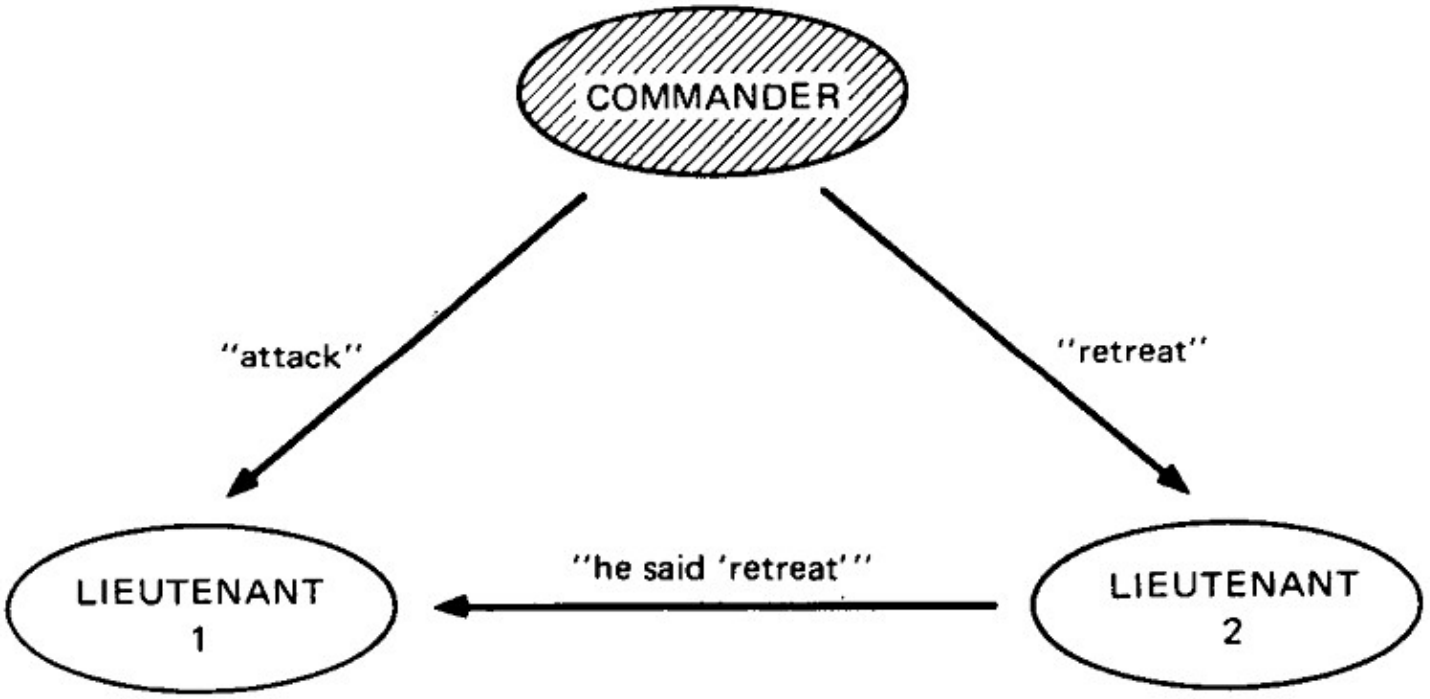


Fig. 2. The commander a traitor.

Chegar a um acordo é complicado!

Detecção de falhas

- Para ter sistemas tolerantes a falhas, devemos primeiramente detectar a falha propriamente dita!

- Duas abordagens:
  - Processos enviam ativamente uns aos outros mensagens “você está vivo?” (*pings*)
  - Processos esperam passivamente por mensagens de processos diferentes (*hellos*)

Detecção de falhas

- Essência: Falhas são detectadas através de temporizadores (***timeouts***)
  - Definir *timeouts* de maneira eficiente é muito difícil e depende da aplicação
  - Difícil distinguir falhas dos processos das falhas da rede
  - Possível solução:
    - Considerar possíveis notificações de falhas entre os membros do sistema
    - *Gossiping*

Estratégias de tolerância a falha

- Falhas de processos
  - **Resiliência de processos**: replicação de processos em grupos

- Falhas de comunicação
  - **Comunicação confiável cliente-servidor**: Canal de comunicação pode exibir falhas por queda, por omissão, arbitrárias
  - **Comunicação confiável de grupo**: Como implementar entrega confiável de mensagens a todos os processos?
  - **Comprometimento distribuído**: Envolve a realização de uma operação por cada membro de um grupo de processos ou por absolutamente nenhum

Comunicação confiável cliente-servidor

- Falhas não atingem somente processos, mas também a comunicação entre eles
  - Falhas por queda
  - Falhas por omissão
  - Falhas de temporização, e
  - Falhas arbitrárias

- Um canal de comunicação confiável deve mascarar falhas por **queda** e **omissão**

Comunicação confiável cliente-servidor

- Comunicação ponto-a-ponto
  - Quando um middleware de comunicação não é usado, a comunicação confiável em um SD pode ser estabelecida com a utilização de um protocolo de transporte confiável
    - Ex: TCP
  - TCP mascara **falhas por omissão**
    - Mensagens perdidas → reconhecimentos e retransmissões

Comunicação confiável cliente-servidor

- Comunicação ponto-a-ponto
  - **Falhas por queda** não são mascaradas
    - Conexão TCP é interrompida abruptamente de modo que nenhuma msg possa ser transmitida pelo canal
  - Um SD pode mascarar tal falha, tentando uma nova conexão, com o envio de uma requisição de conexão

Comunicação confiável cliente-servidor

- Utilizando Middleware de Comunicação: RPC
  - Objetivo do RPC é ocultar comunicação fazendo chamadas de procedimentos remotos parecerem chamadas locais
  - Quando erros ocorrem, é difícil mascarar a diferença entre chamadas locais e remotas
    - Ações a serem tomadas podem ser bem diferentes
  - Vamos considerar algumas categorias de erros e possíveis soluções

Comunicação confiável cliente-servidor

- Diferentes classes:
  1. Cliente não consegue localizar o servidor
  2. A mensagem de requisição do cliente para o servidor se perde
  3. O servidor cai após receber uma requisição
  4. A mensagem de resposta do servidor para o cliente se perde
  5. O cliente cai após enviar uma requisição

Comunicação confiável cliente-servidor

- **Cliente não consegue localizar o servidor**
  - Possíveis problemas:
    1. Servidores podem ter caído
    2. Cliente pode ter sido compilado usando uma versão de apêndice antiga
  - Solução:
    - Ativar uma exceção/sinalização → acarreta em perda da transparência

Comunicação confiável cliente-servidor

- **Mensagens de requisição perdidas**
  - Rede subjacente não é confiável
  - Solução:
    - Temporizador de espera para a mensagem de resposta servidor → deverá reconhecer a mensagem original da mensagem de retransmissão

Comunicação confiável cliente-servidor

- **Quedas de servidor**

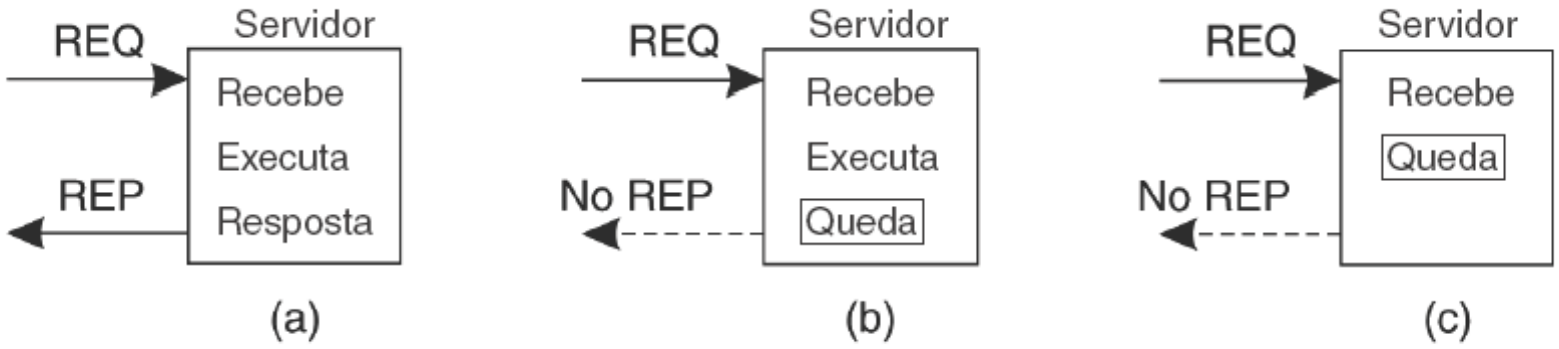


Figura 8.6 Servidor em comunicação cliente–servidor. (a) Caso normal. (b) Queda após a execução. (c) Queda antes da execução.

Comunicação confiável cliente-servidor

- **Mensagens de respostas perdidas**
  - Problema: cliente não sabe com certeza por que não houve reposta
    - Requisição ou reposta que se perdeu ou servidor está lento?
  - No caso de requisição **idempotente**: operações que podem ser repetidas sem causar nenhum dano
    - Ex: leitura de arquivos → retransmissão pode ser feita sem problemas a consistência do SD

Comunicação confiável cliente-servidor

- **Mensagens de respostas perdidas**

- Possíveis soluções:
  - Estruturar todas as requisições como idempotentes
  - Abordagem TCP-like: cliente designa um número de sequência a cada requisição → servidor mantem informações de cada cliente

Comunicação confiável de grupo

- **Comunicação Multicast** → mensagens são entregues a um grupo de processos
- Na prática é uma tarefa complicada!
  - Camada de transporte oferece apenas comunicação ponto-a-ponto confiável (TCP)

- Possível solução → multicast em camada de aplicação
  - Estabelecer comunicações ponto-a-ponto entre os processos
  - Muito usado na prática
  - Problema: Desperdício de largura de banda da rede

Comunicação confiável de grupo

- De fato, o que é multicast confiável?
  - Significa que uma mensagem enviada a um grupo de processos deve ser entregue a cada membro do grupo

- O que ocorre se durante a comunicação um processo se juntar ao grupo?
- O que ocorre se um processo sair deste grupo?

Comunicação confiável de grupo

- Para apresentar soluções vamos considerar dois cenários:
  1. Processos estão funcionando corretamente e o grupo é estático
  2. Processos falham!

Comunicação confiável de grupo

- **Cenário 1: premissas**
  - Consideremos o caso em que um único remetente queira enviar uma mensagem multicast a vários receptores
    - Ex: Algoritmo de Berkeley para sincronizar os relógios
  - Rede de comunicação não é confiável: mensagem multicast pode se perder em algum ponto do caminho e ser entregue a alguns, mas não a todos os receptores

Comunicação confiável de grupo

- **Cenário 1: premissas**
  - Processos não falham e não se juntam ao grupo nem saem dele enquanto a comunicação está em curso
  - Multicast confiável → Toda mensagem deve ser entregue a cada membro do grupo no momento em questão

Comunicação confiável de grupo

- **Cenário 1: Solução 1**

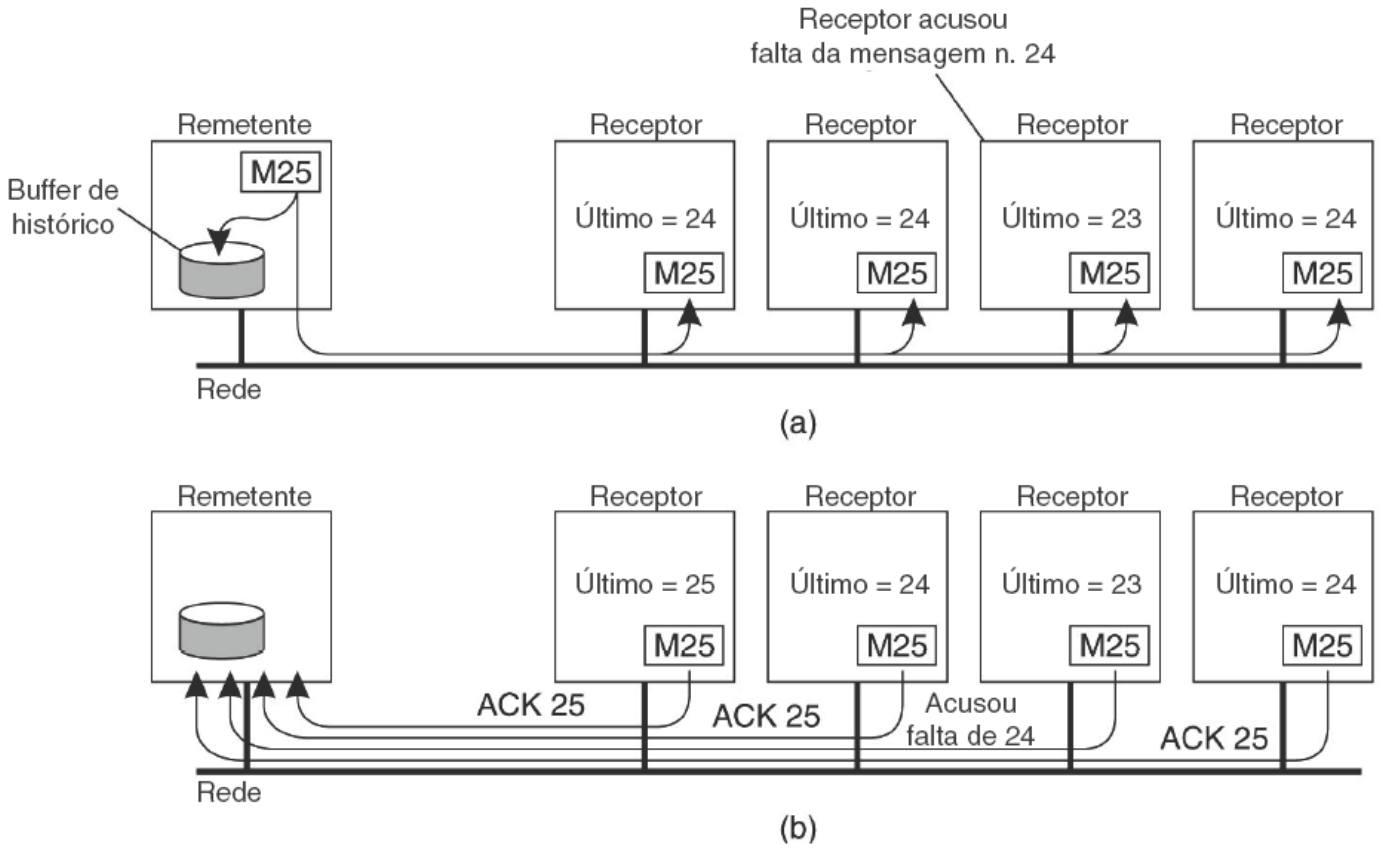


Figura 8.8 *Uma solução simples para multicast confiável quando todos os receptores são conhecidos; a premissa é que nenhum falhe. (a) Transmissão de mensagem. (b) Realimentação de relatório.*

Comunicação confiável de grupo

- **Cenário 1: Solução 1**
  1. Processo remetente designa um número de sequência a cada mensagem multicast
  2. Mensagens são recebidas na ordem
  3. Cada mensagem multicast é armazenada no remetente, em um *buffer*
  4. Mensagem é mantida até receptores confirmem o recebimento (ACK)
  5. Retransmissão em caso de reconhecimento negativo (NACK) ou *timeout*

Comunicação confiável de grupo

- **Cenário 1: Solução 1 (Questões...)**
  - Como reduzir o número de mensagens retornadas ao remetente? **piggybacking**?
  - Como fazer a retransmissão? Ponto-a-ponto ou novamente multicast?
  - E a escalabilidade?
    - Com N receptores, o remetente deve estar preparado para aceitar no mínimo N ACKs → **implosão de retorno**

Comunicação confiável de grupo

- **Cenário 1: Solução 2**
  - Receptores enviam mensagem de retorno somente para informar ao remetente a FALTA de uma mensagem (NACK)
  - Retornar somente reconhecimentos negativos melhora a escalabilidade [Towsley *et al.* 1997], mas não garante que implosões de retorno nunca acontecerão
  - Problema: Remetente será forçado a manter uma mensagem em seu buffer de histórico para “sempre” → uso de timeout para retirada da msg pode levar ao caso onde uma requisição pode não ser efetivada!

Comunicação confiável de grupo

- **Cenário 1: Solução 3** [Floyd *et al.* 1997]
  - Objetivo principal: Reduzir mensagens de retorno (**supressão de retorno**)
  - Utiliza o protocolo de multicast confiável escalável (SRM)
  - Somente reconhecimentos negativos (NACK) são devolvidos como realimentação. Aplicação decide como detectar a perda de uma mensagem
  - Ao reconhecer que está faltando uma mensagem, receptor envia o pedido da mensagem perdida, **usando multicast**, ao resto do grupo

Comunicação confiável de grupo

- **Cenário 1: Solução 3** [Floyd *et al.* 1997]
  - Utilizar realimentação multicast permite que outro membro do grupo suprima seu reconhecimento!
  - Exemplo:
    - Suponha que vários receptores tenham percebido a falta da msg **m**
    - Cada um deles precisará retornar um reconhecimento negativo ao remetente **S**, para retransmissão de **m**
    - Se considerarmos que os reconhecimentos são enviados em multicast, basta uma única mensagem de requisição para que o pedido chegue até **S**

Comunicação confiável de grupo

- **Cenário 1: Solução 3** [Floyd *et al.* 1997]
  - Utilizar realimentação multicast permite que outro membro do grupo suprima seu reconhecimento!
  - Exemplo:
    - Um receptor **R** que não recebeu a msg **m** escalona uma msg de realimentação com certo atraso aleatório
    - A requisição para retransmissão não é enviada até passar algum tempo aleatório
    - Se um requisição de **m** chegar a **R,** antes do timeout, **R** não enviará a msg de realimentação negativa
    - Espera-se que somente uma msg de pedido de retransmissão de **m** chegue a **S**

Comunicação confiável de grupo

- **Cenário 1: Solução 3** [Floyd *et al.* 1997]

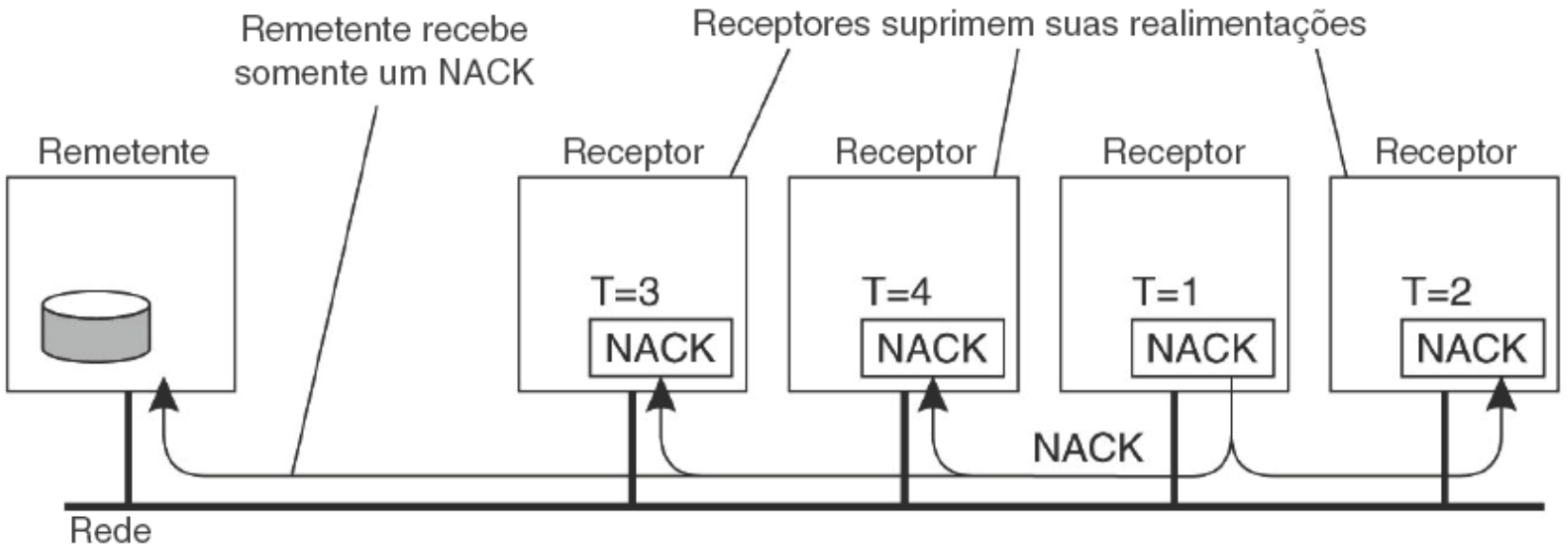


Figura 8.9 Vários receptores escalonaram uma requisição para retransmissão, mas a primeira requisição de retransmissão resulta na supressão de outras.

Comunicação confiável de grupo

- **Cenário 1: Solução 3** [Floyd *et al.* 1997]
- Aprimoramento:
  - Permitir que um receptor tenha o papel de transmitir uma mensagem m, mesmo antes de a requisição de retransmissão chegar ao remetente original
- Exemplo de aplicação:
  - Aplicações colaborativas, como a de *whiteboard* compartilhado

Comunicação confiável de grupo

- **Cenário 1: Solução 3** [Floyd *et al.* 1997]
  - Problemas
    - Garantir que somente uma requisição de retransmissão chegue ao remetente **S** → escolha dos temporizadores!
    - Interrupção dos processos os quais a msg já foi entregue → separar os processos que não receberam **m** em um grupo separado [Kasera *et al.* 1997] → gerenciamento dos grupos!
    - Reunir os processos em grupos que tendem a perder mensagens [Lui *et al.* 1998]

Comunicação confiável de grupo

- **Cenário 1: Solução 4:** Controle de Realimentação Hierárquico
  - Grupo é dividido em sub-grupos, organizados em árvore
  - Cada sub-grupo possui um coordenador que gerencia os pedidos de retransmissão
  - O coordenador possui um buffer para armazenar as msgs para atender pedidos dos membros do seu sub-grupo
  - Problema: Manutenção da árvore

Comunicação confiável de grupo

- **Cenário 1: Solução 4:** Controle de Realimentação Hierárquico

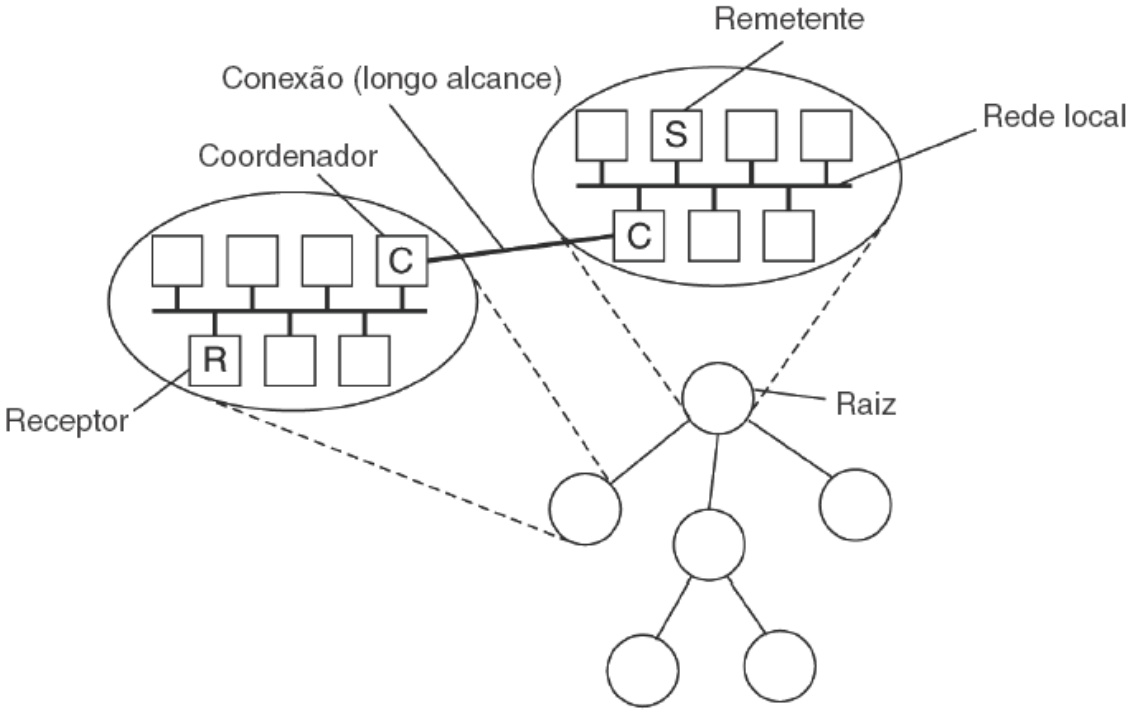


Figura 8.10 Essência do multicast confiável hierárquico. Cada coordenador local repassa a mensagem a seus filhos e mais tarde manipula requisições de retransmissão.