

Sistemas Distribuídos e Redes de Sensores

Comunicação em Grupo

abril de 2013

exemplos:

- replicação de serviços
- clientes com estado compartilhado
- computação científica
- ...

Primitiva de comunicação em grupo

- um processo envia uma mensagem para um grupo de processos e *todos* os destinatários recebem
- garantias variadas

Broadcast ou Multicast

- em redes: conceito de entrega *para todos* e *para alguns*
- conceitos se confundem em sistemas distribuídos

Comunicação em Grupo: Problemas Novos

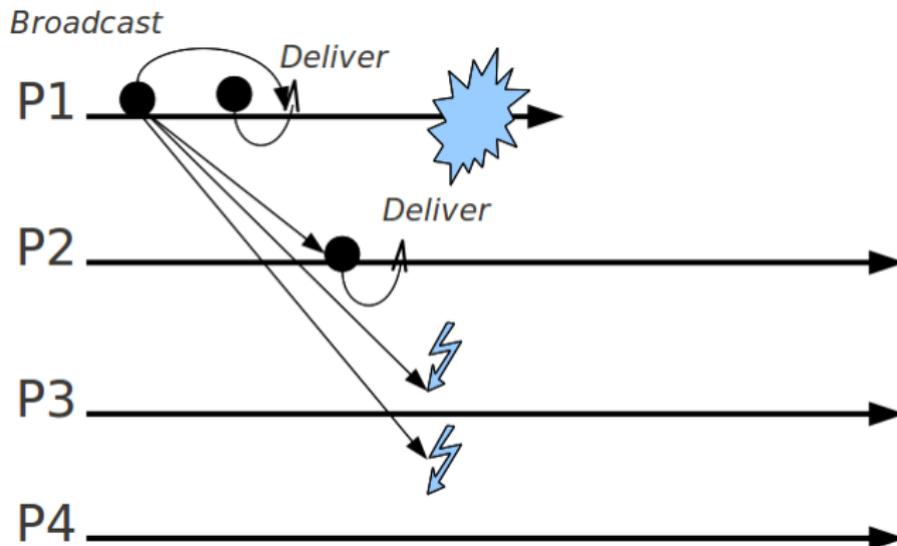
- alguns recebem e outros não
- processos recebem mensagens em ordens diferentes

- confiabilidades
- ordenação

Implementação

- chegada X entrega

Exemplo: envio de msg para um grupo de processos

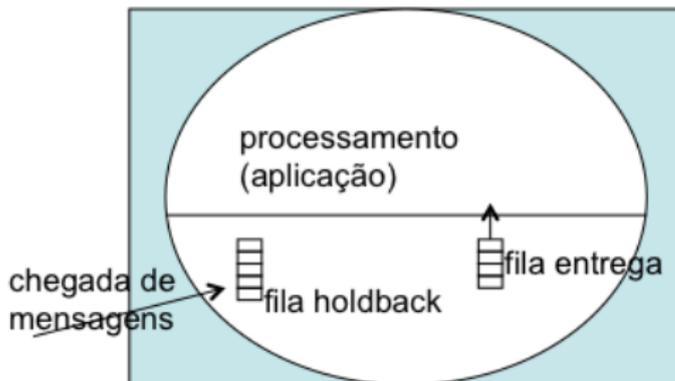


Níveis de garantia de entrega

- 1 *Melhor-esforço (best-effort)*: garantia de entrega entre processos corretos
- 2 *Confiável (reliable)*: garantia *all-or-nothing* mesmo se o emissor falhar
- 3 *Ordem xyz*: garantia de entrega na ordem *xyz*

Garantia de entrega: Implementação

- uso de *holdback queue*
- em geral, mensagens com *identificadores únicos*
 - custos de espaço e processamento!



Garantia de entrega: Implementação

- Um processo envia uma msg em um passo de comunicação para todos os processos do sistema, incluindo ele mesmo
- O custo para garantir confiabilidade é apenas do lado do emissor (se ele falhar, nenhuma garantia de entrega é oferecida)

Interface e propriedades do “Bcast melhor-esforço”

Molule:

Name: BestEffortBroadcast (beb).

Events:

Request: <bebBroadcast | m>: bcast m

Indication: <bebDeliver | src, m>: entrega m

Properties:

BEB1: Validade melhor-esforço: se P_i e P_j
são corretos, toda msg de P_i é entregue por P_j

BEB2: Não duplicação: nenhuma msg é entregue
mais de uma vez

BEB3: Não-criação: se a msg é entregue por P_j
então ela foi difundida por P_i

Algoritmo Bcast básico

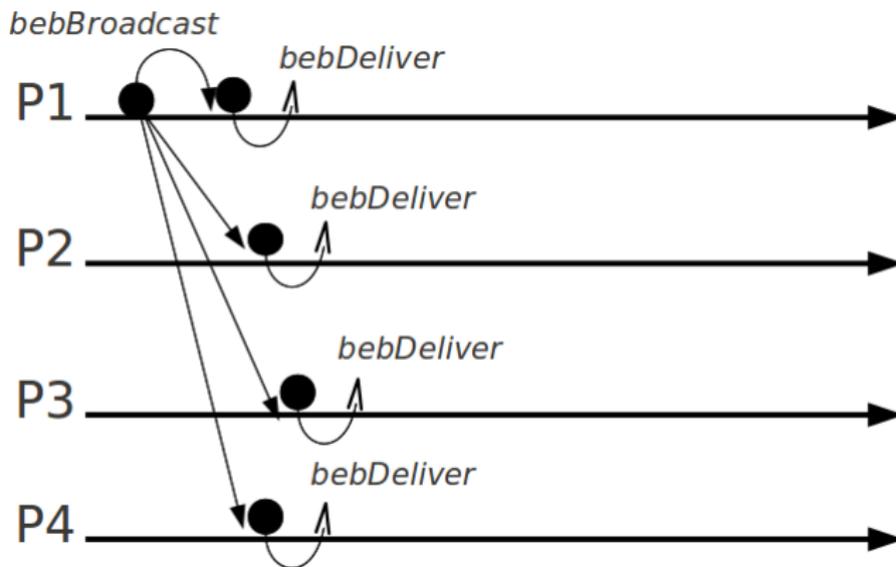
```
Implements: BestEffortBroadcast (beb)
```

```
Uses: PerfectPointToPointLinks (pp2p)
```

```
event <bebBroadcast | m> do  
  forall pi do  
    trigger <pp2pSend | pi, m>;
```

```
event <pp2pDeliver> | pi, m> do  
  trigger <bebDeliver | pi, m>;
```

Exemplo de execução de Bcast básico



Desempenho: o algoritmo requer um único passo de comunicação e troca N mensagens

Interface do enlace

Module:

Name: PerfectP2PLink

Events:

Request: < Send | dest, msg >

Indication: < Deliver | src, msg >

- entrega confiável: se p_i manda para p_j e nenhum deles falha, p_j em algum momento recebe
- ausência de duplicação de mensagens
- ausência de criação de mensagens

Propriedades do enlace

propriedades caras se ambiente é de redes de sensores!

- 1 interface *FairLossP2Plinks*
- 2 interface *StubbornP2Plinks*: retransmite eternamente (didático mas sem bom desempenho...)
- 3 interface *PerfectP2Plinks*: elimina duplicatas
- 4 entrega mesmo com falhas intermitentes...

- 1 se transmissor não falhar, garantia de entrega a todos
- 2 se transmissor falhar:
 - processos podem “discordar” sobre entrega de mensagem
 - broadcast de melhor esforço pode não ter transmitido para todos ou falha pode ter ocorrido antes de *StubbornP2PLinks* ter feito todas as retransmissões necessárias

- todos os processos devem receber (tratar) o mesmo conjunto de mensagens
 - noção de *acordo*
- solução para modelo *fail-stop*

Interface e propriedades do *Bcast confiável*

Molule:

Name: Reliable Broadcast (rb).

Events:

Request: <rbBroadcast | m>: bcast m

Indication: <rbDeliver | src, m>: entrega m

Properties:

RB1: Validade: se P_i e P_j

são corretos, toda msg de P_i é entregue por P_j

RB2: Não duplicação: nenhuma msg é entregue
mais de uma vez

RB3: Não-criação: se a msg é entregue por P_j
então ela foi difundida por P_i

RB4: Acordo: Se msg é entregue por processo correto p_i ,
então m é entregue em algum momento para qualquer
processo correto p_j

“Pior-caso”, o emissor sempre falha

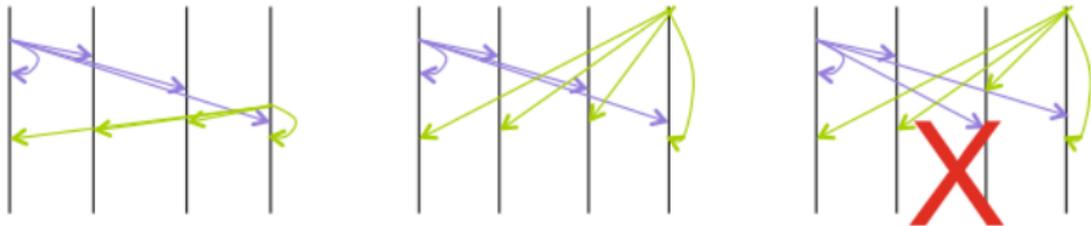
```
Implements: ReliableBroadcast (rb)
Uses: BestEffortBroadcast (beb)
event <Init> do
  delivered := 0;

event <rbBroadcast | m> do
  delivered := delivered U {m};
  trigger <rbDeliver | self, m>;
  trigger <bebBroadcast | [DATA,self,m]>;

event <bebDeliver> | pi, [DATA,self,m] > do
  if(m não pertence a delivered) do
    delivered := delivered U {m};
    trigger <rbDeliver | source_m, m>;
    trigger <bebBroadcast | [DATA,source_m,m]>;
```


- As soluções anteriores não garantem ordem de entrega das mensagens enviadas por processos diferentes (**mensagens de um mesmo processo devem ser entregues na ordem em que foram difundidas**)
- Algumas aplicações precisam de garantias de ordem de entrega.
 - ordem total
 - ordem causal

Ordem Total



- Não importa a ordem de entrega, mas deve ser a mesma em todos os processos do grupo.

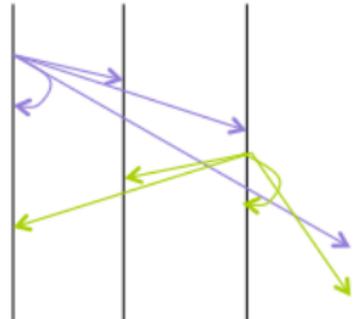
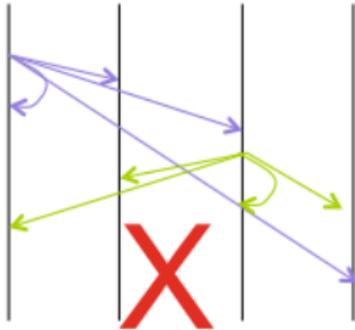
Ordem total com sequenciador

Ordem total com votação de ordem

Ordem Causal

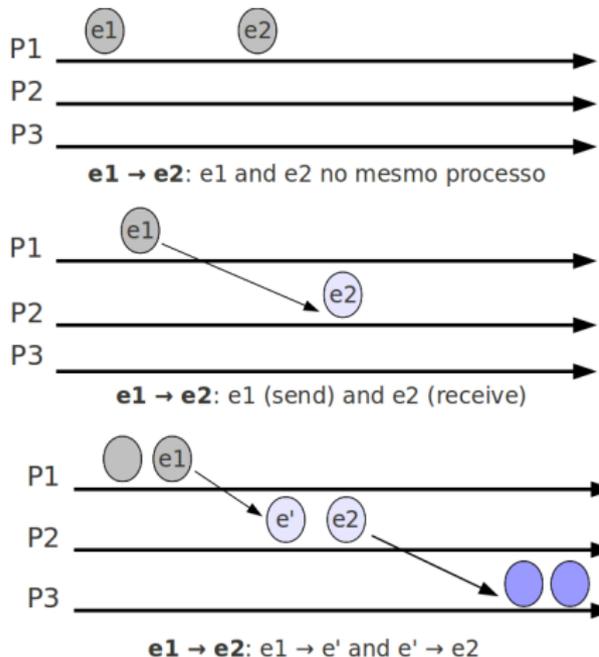
possível relação de causalidade

- uma mensagem pode ter sido consequencia de outra ainda não vista



Causalidade em eventos distribuídos...

- $a \rightarrow b$: o evento a ocorre antes do evento b se
 - a ocorre antes de b em um mesmo processo P
 - $a = \text{send}(m)$ no processo P e $b = \text{receive}(m)$ no processo Q
 - existe c tal que a precede c e c precede b
- $a \parallel b$: os eventos a e b são concorrentes



Interface e propriedades do “Bcast ordem causal”

Molule:

Name: ReliableCausalOrder (rco)

Events:

Request: <rcoBroadcast | m>: bcast m

Indication: <rcoDeliver | src, m>: entrega m

Properties:

CB: Entrega causal: m2 só é entregue por Pi
se toda msg mi tal que mi->m2 foram entregues

RB1: Validade

RB2: Não-duplicação

RB3: não-criação

RB4: Acordo

Algoritmo Bcast ordem causal

```
Uses: ReliableBroadcast (rb)
event <Init> do
  delivered := 0; past := 0;
event <rcoBroadcast | m> do
  trigger <rbBroadcast | [DATA,past,m]>;
  past := past U {(self,m)};
event <rbDeliver> | pi, [DATA,past_m,m] > do
  if(m não-pertence a delivered) then
    forall (s_n, n) in past_m do
      if(n não-pertence delivered) then
        trigger <rcoDeliver | s_n, n>;
        delivered := delivered U {n};
        past := past U {(s_n,n)};
      trigger <rcoDeliver | pi, m>;
      delivered := delivered U {m};
      past := past U {(pi,m)};
```

custos proibitivos... cada mensagem carrega todas as que a precedem causalmente

Broadcast causal com espera

- uso de relógio lógico e timestamps: funciona se tempo de entrega de mensagens tem limite máximo

Vetor de timestamps: idéia básica

- Dados N processos, usa-se um vetor de N elementos (ao invés de um valor escalar)
- A cada evento “e”, associa-se um **vetor de timestamps** cujo i -ésimo elemento indica quantos mensagens do processo i já foram vistas

Definição da relação $<$

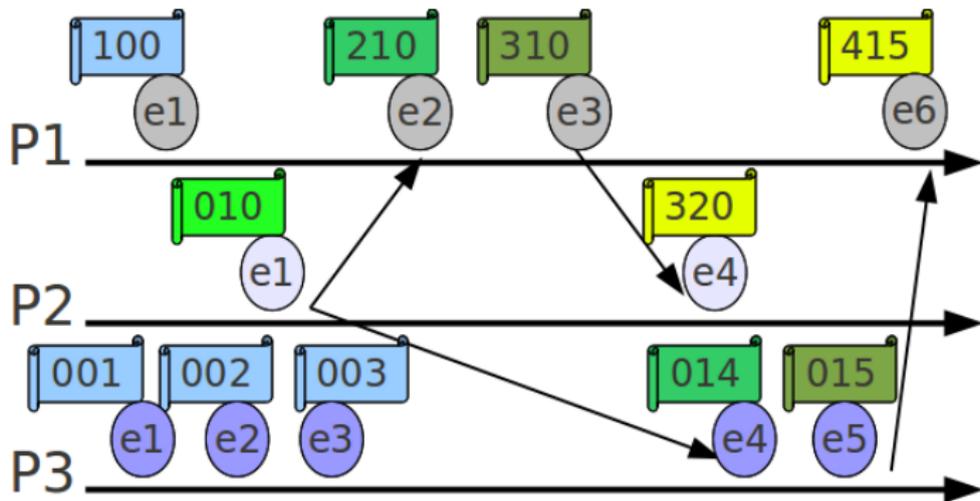
- Sejam a e b dois eventos quaisquer
- $VT(a) < VT(b)$:
 - 1 $\forall i, VT(a)[i] \leq VT(b)[i]$
 - 2 $\exists j, VT(a)[j] < VT(b)[j]$
- Note que $<$ é uma *ordem parcial*, existem eventos que não podem ser comparados por serem concorrentes
 - Ex., $x = [0, 0, 1, 0]$ e $y = [1, 0, 0, 0]$, pois $VT(x) \neq VT(y)$ e não vale $VT(x) < VT(y)$ nem $VT(y) < VT(x)$

Funcionamento

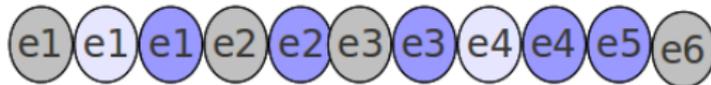
- Cada processo mantém seu VT que começa com 0 em todas as posições
- Para cada evento “e” local e de envio de msg, incrementa o componente do processo local no VT e associa VT_p ao evento “e”: $VT_p[p] = VT_p[p] + 1$ e $VT(e) = VT_p$
- Quando Q recebe uma mensagem, atualiza cada campo do seu VT:
 - $VT_q[i] = VT_q[i] + 1, i = q$
 - $VT_q[i] = \max(VT_q[i], VT_m[i]), i \neq q$

Esta última parte garante que tudo que acontecer depois em P_q passa a ser causalmente relacionado com tudo o que aconteceu antes em P_i

Exemplo de uso de VT para ordenação de eventos



Ordem causal por relógio de Lamport



- é necessário não apenas estabelecer ordem mas garantir que foram vistas todas as mensagens que precedem causalmente determinada mensagem!

Exemplo de entrega em ordem causal

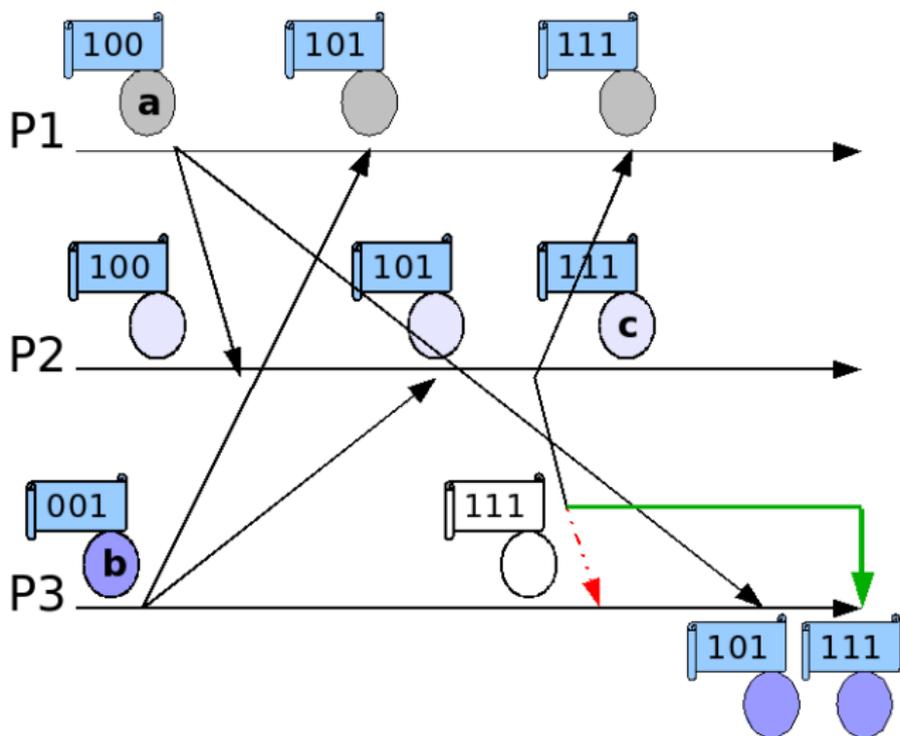


Figura: A entrega de **c** deve ser postergada até a chegada de **a**

- R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming* Springer, 2006
- material aulas Silvana Rossetto, 2012.