Um Algoritmo Probabilista de Recuperação de Erros para Difusão Fiável

Zhen Xiao, Kennneth P. Birman

Apresentação: Henrique Moniz

Sumário

- Introdução: multicast e o contexto do problema
- Tipos de multicast e multicast fiável
- A proposta: Randomized Reliable Multicast Protocol
  - Descrição do funcionamento
  - Simulação e análise de performance
  - Conclusões

Introdução: o que é Multicast?

- Unicast: um emissor transmite para um receptor (ex: TCP)

- Multicast: um emissor transmite para multiplos receptores
  - Maior eficiência na distribuição de dados: as mensagens são apenas enviadas uma vez e vão sendo criadas cópias à medida que os links para os receptores se vão separando

IP Multicast

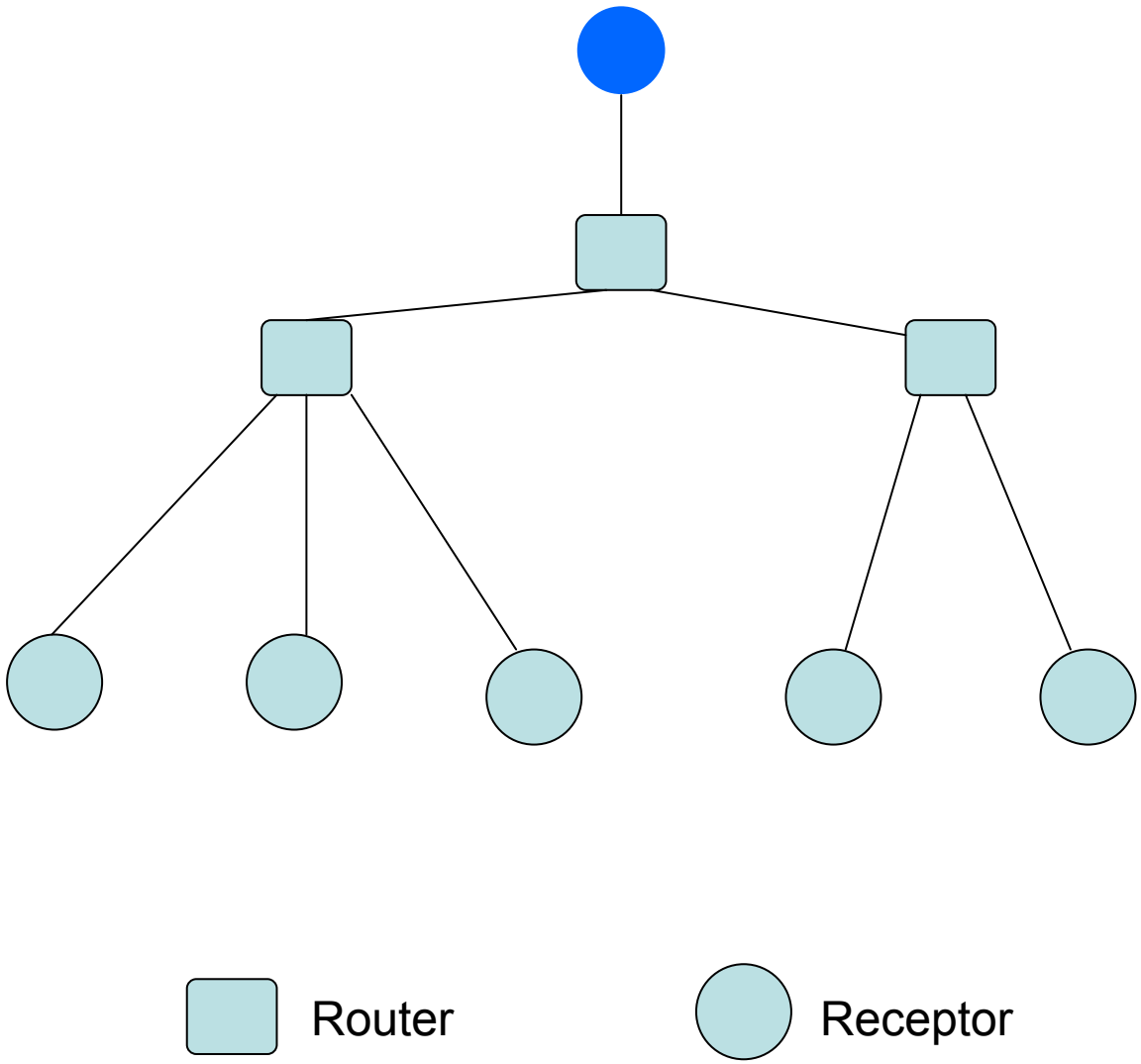
- Não-fiável: as mensagens transmitidas são “best-effort”
- Existe a necessidade de proporcionar um serviço de multicast fiável a muitas aplicações
- O grande problema: como consegui-lo de forma eficiente e escalável?

Multicast Fiável

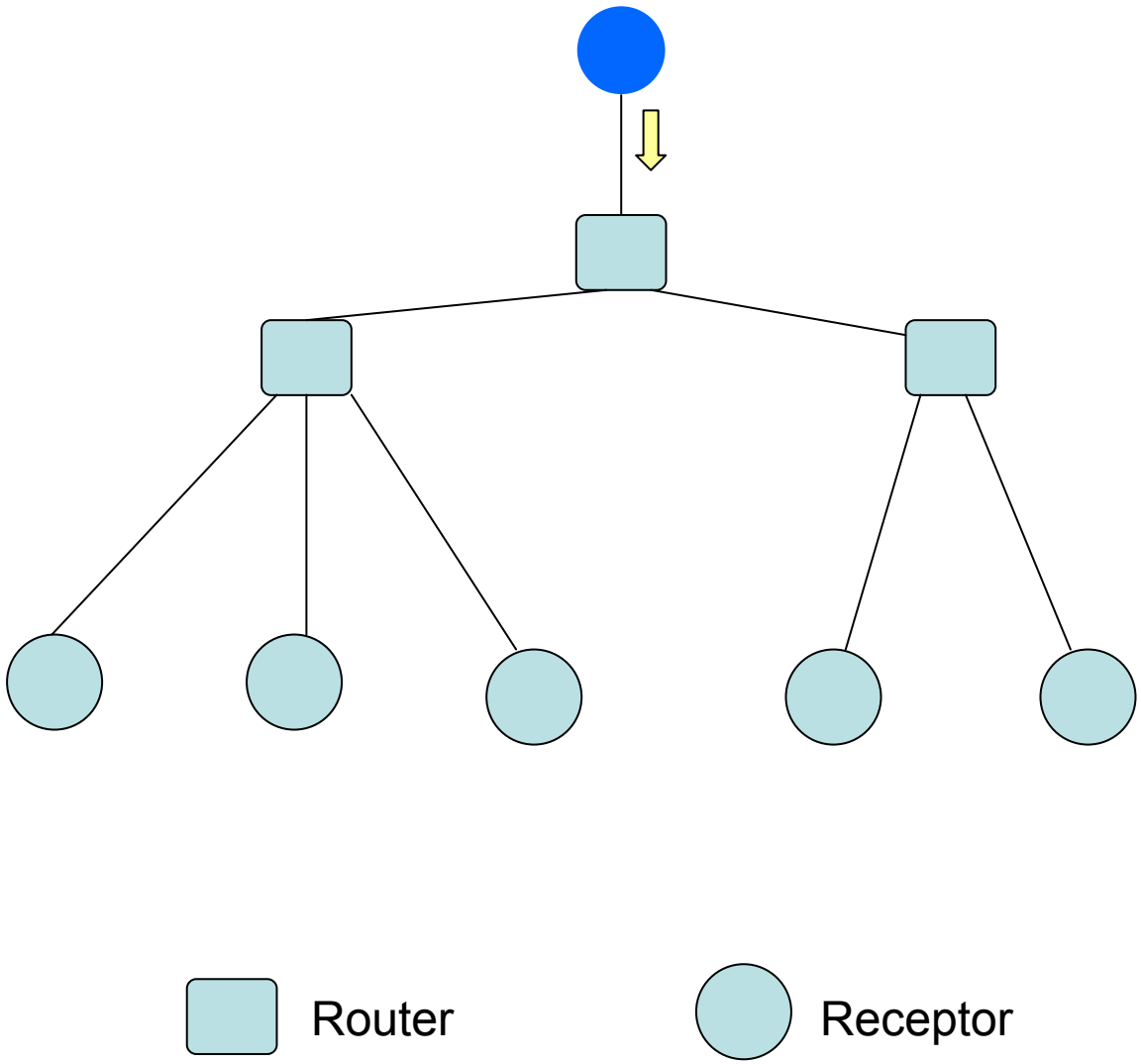
- Modelo simplista: o emissor é responsável pela entrega fiável das mensagens

- Fenómemo de “ack implosion”: uma tempestade de acks ou nacks oriunda de um grande grupo de receptores coloca um grande fardo do emissor

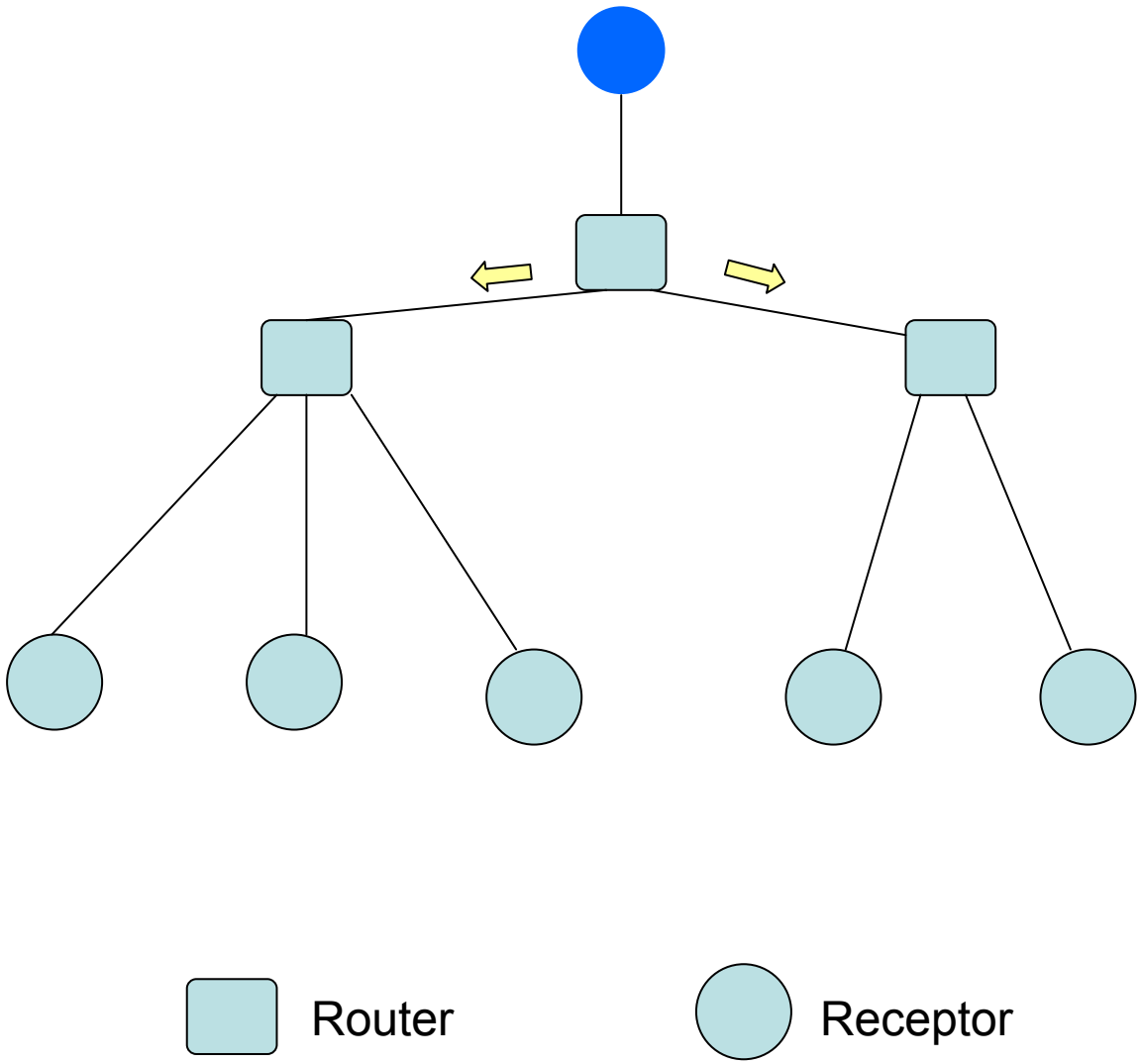
ACK Implosion



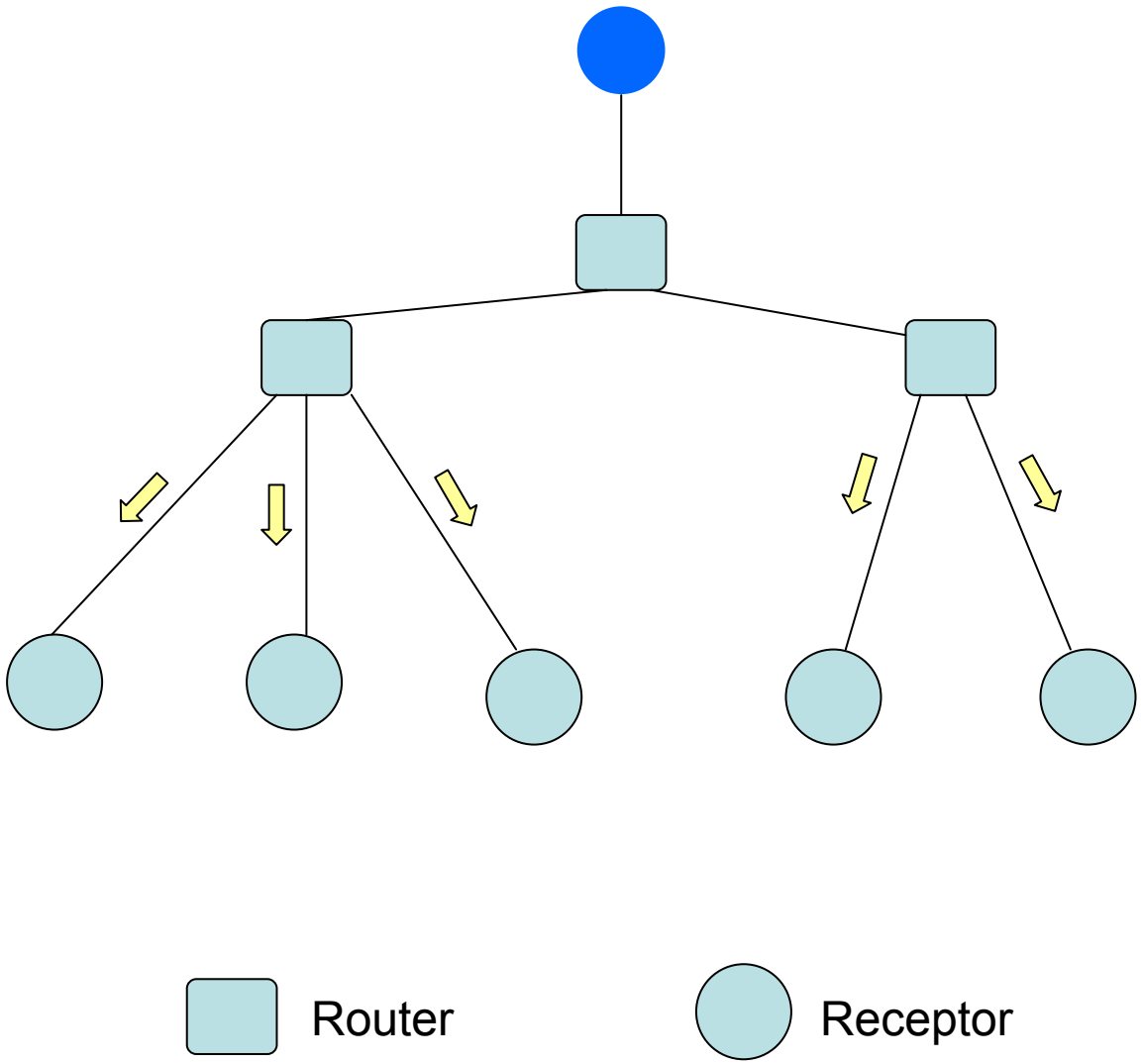
ACK Implosion



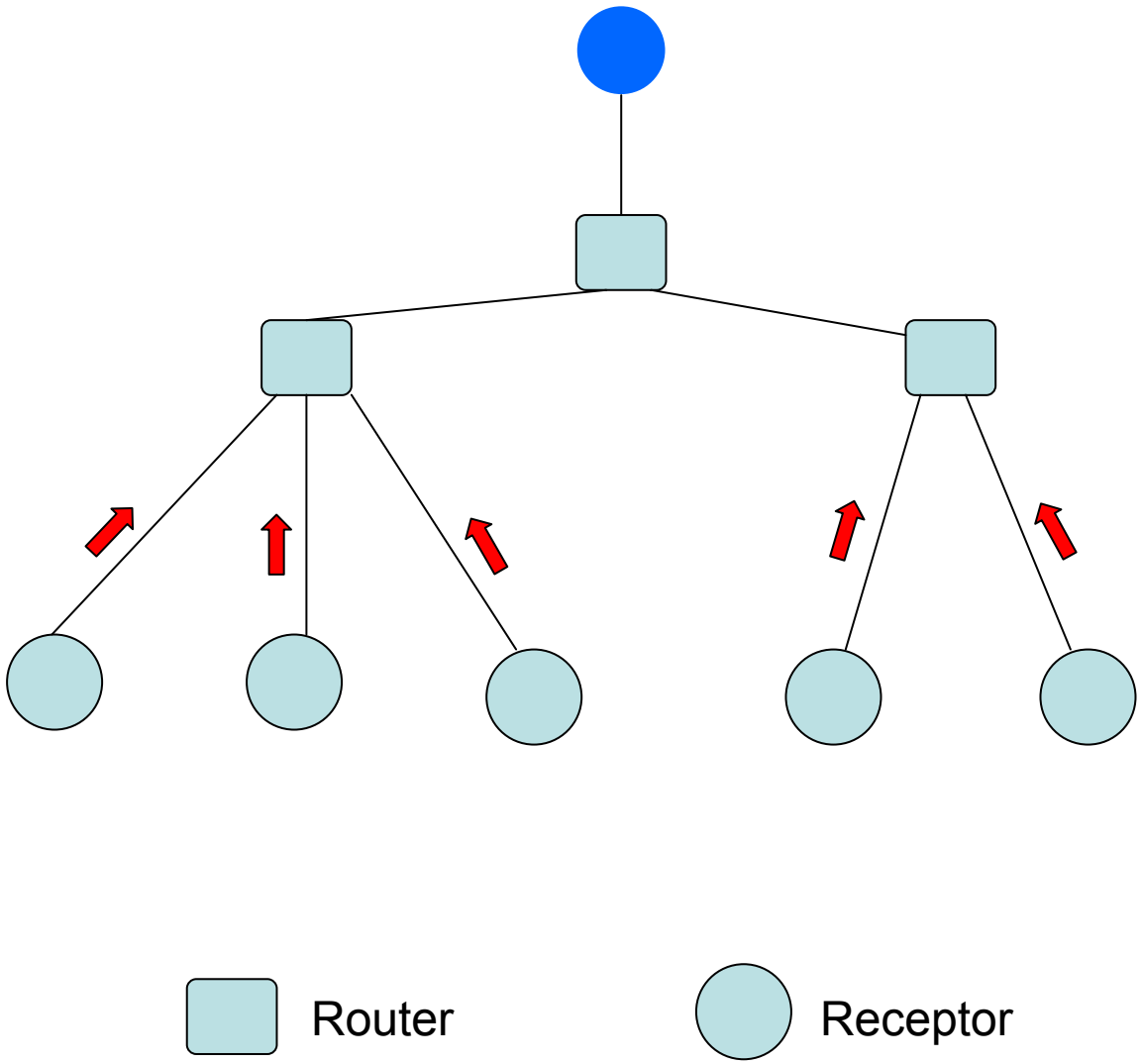
ACK Implosion



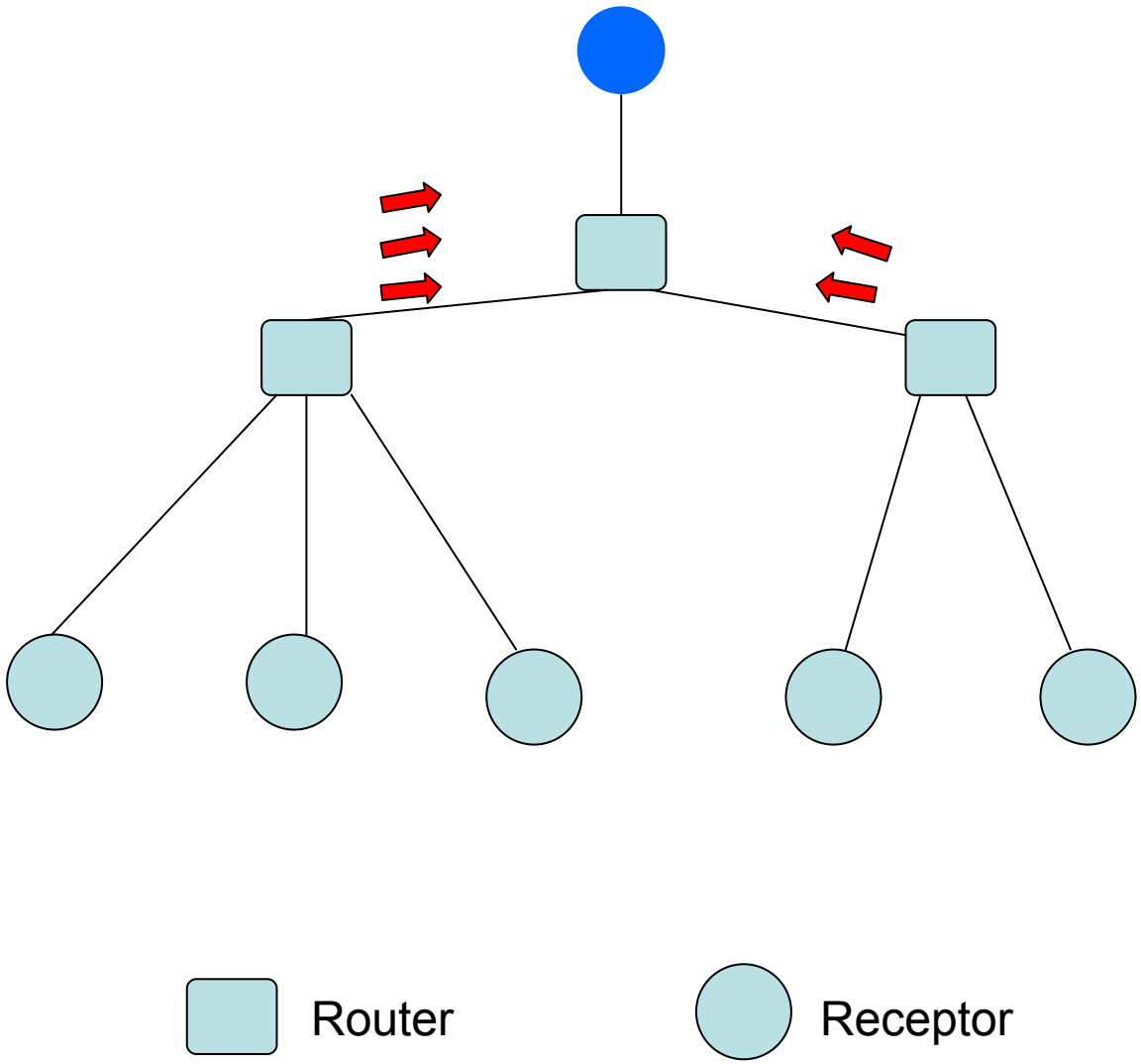
ACK Implosion



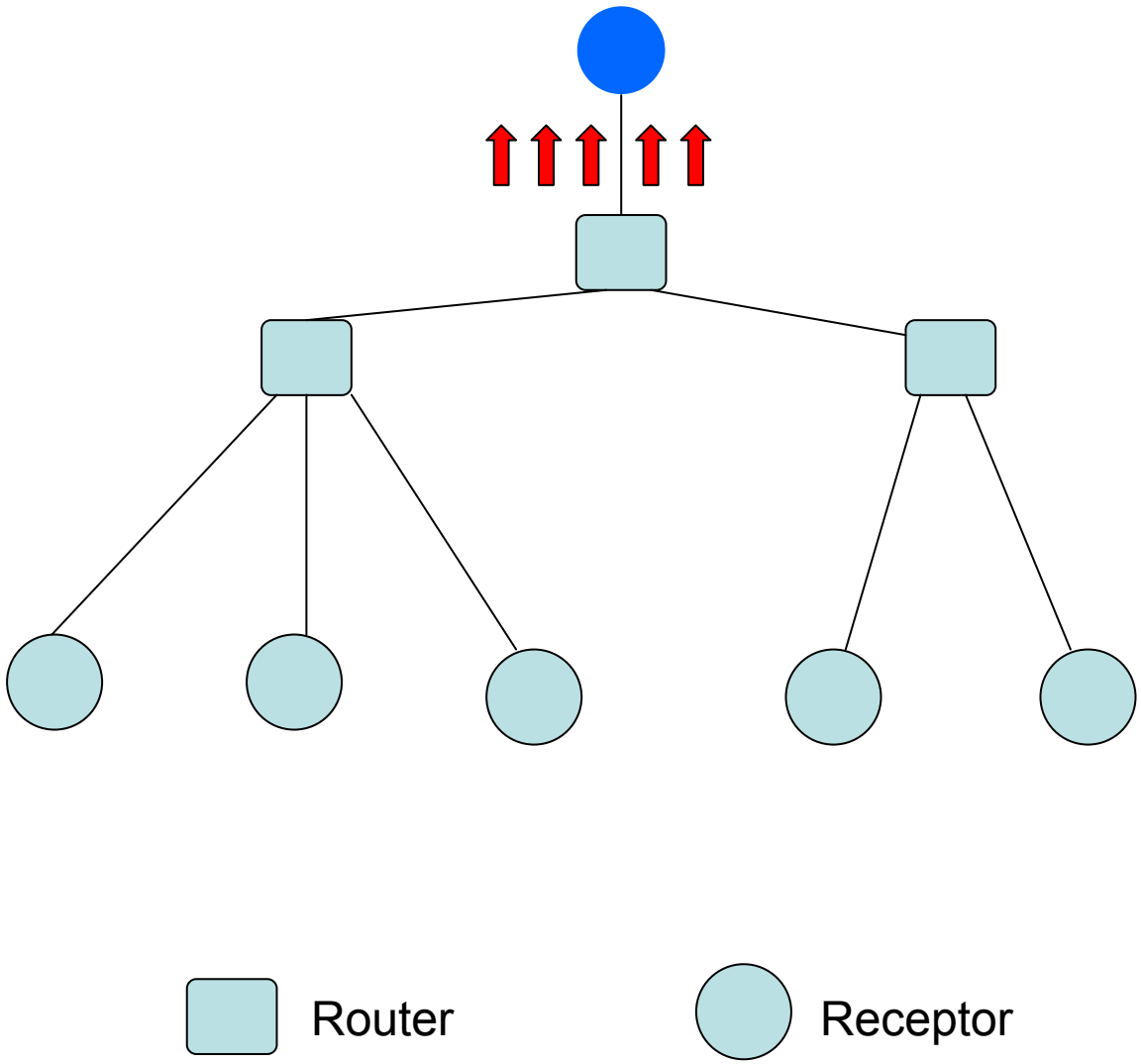
ACK Implosion



ACK Implosion



ACK Implosion



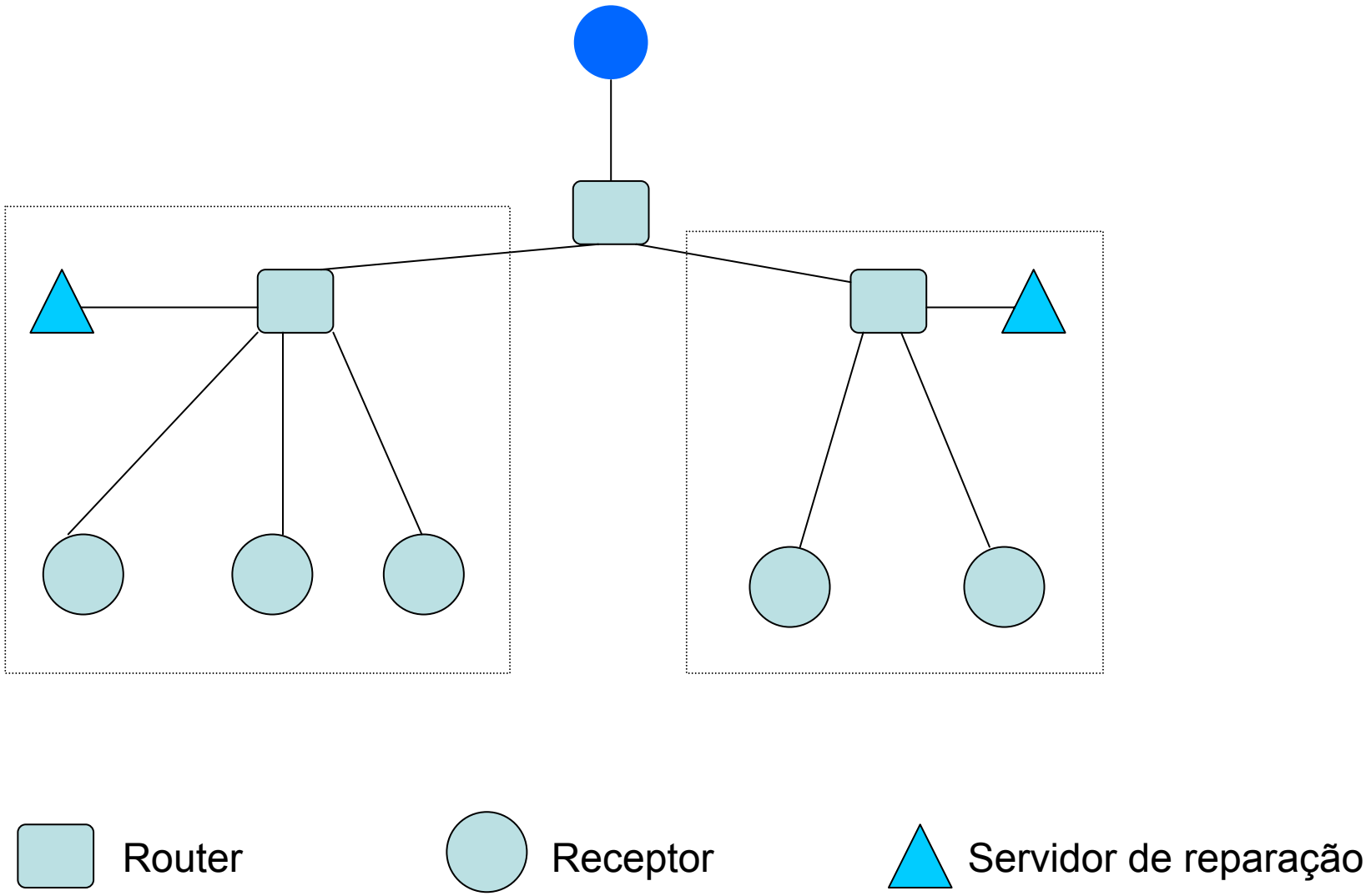
Multicast fiável

- O receptor é o responsável pela recepção fiável das mensagens
- Maior escalabilidade ainda que limitada: o emissor já não é tão sobrecarregado
- Exemplo: Scalable Multicast Protocol (SRM)

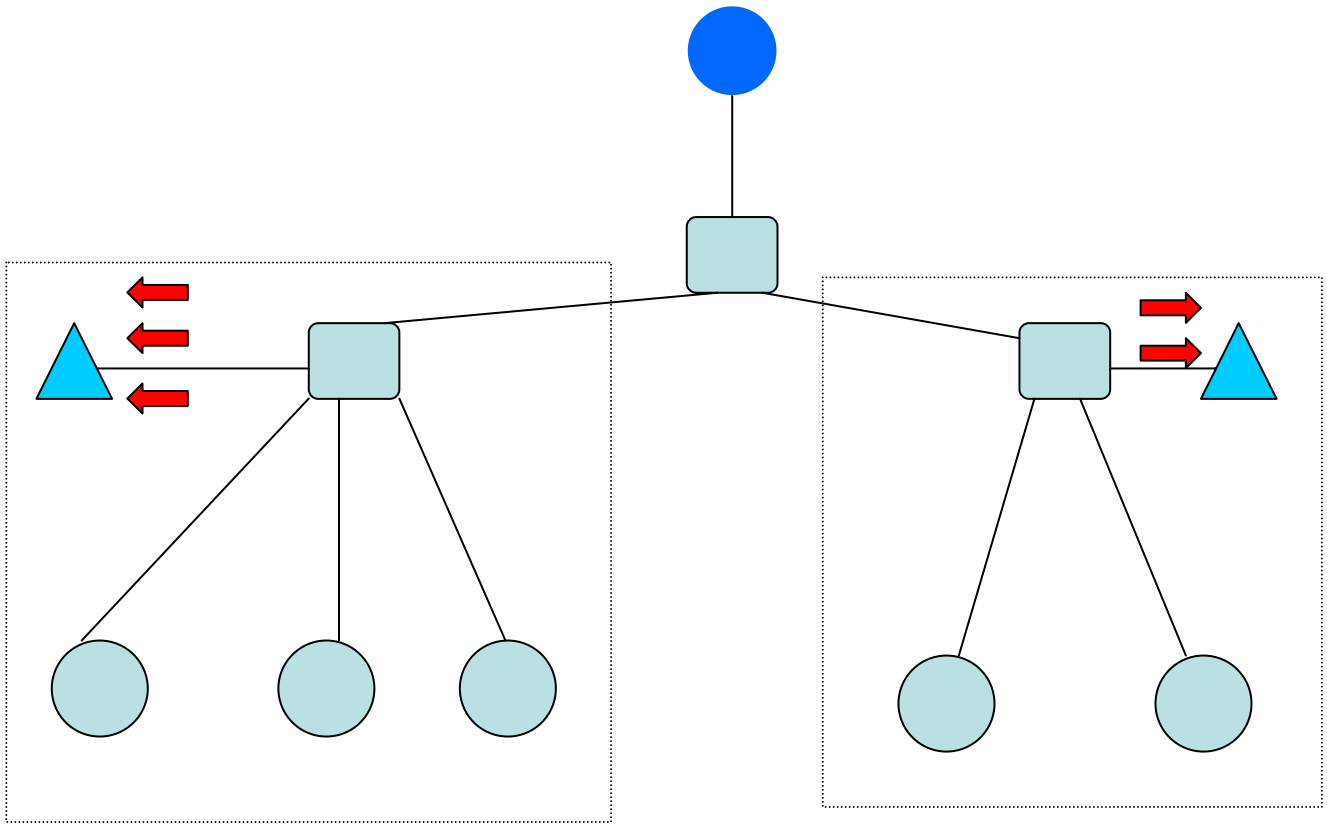
Multicast Fiável

- Protocolos baseados em árvore: para aplicações com apenas um emissor
  - Os receptores são agrupados em regiões baseadas na sua proximidade geográfica
  - Para cada região existe um servidor de reparação que é responsável pela recuperação de erros ao nivel dessa região
    - Evita implosão de mensagens no emissor
    - Continuam a existir problemas de implosão a nivel regional
    - Os servidores de reparação representam pontos únicos de falha

Tree-Based



Tree-Based

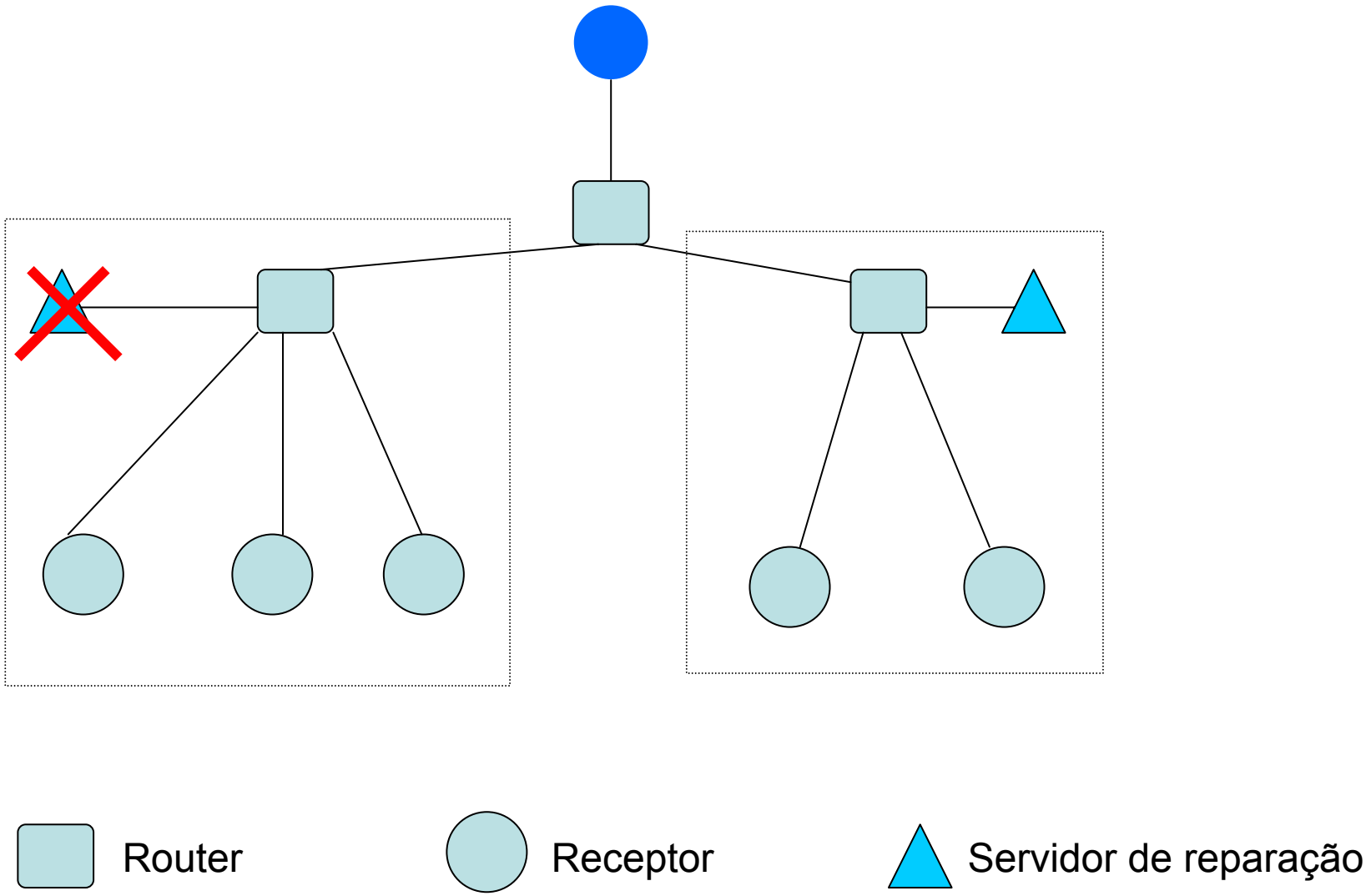


Router

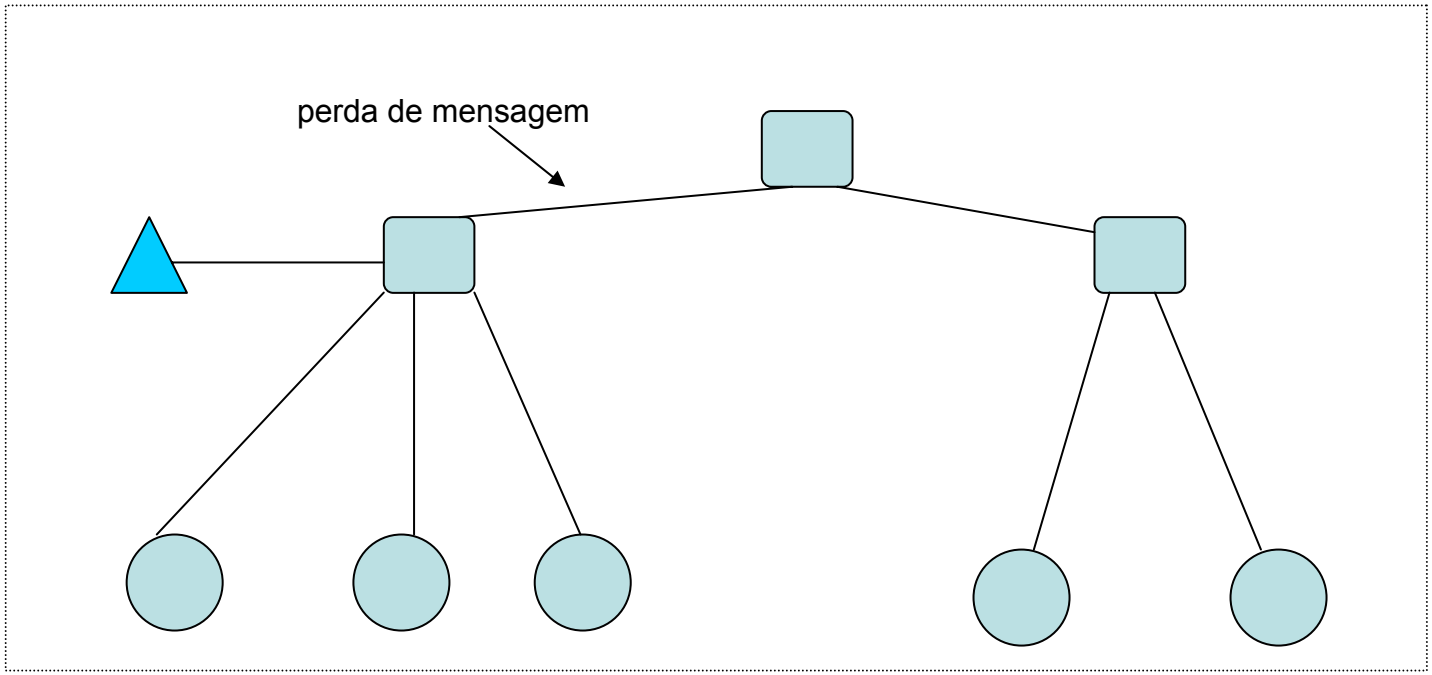
Receptor

Servidor de reparação

Tree-Based



Tree-Based



Router

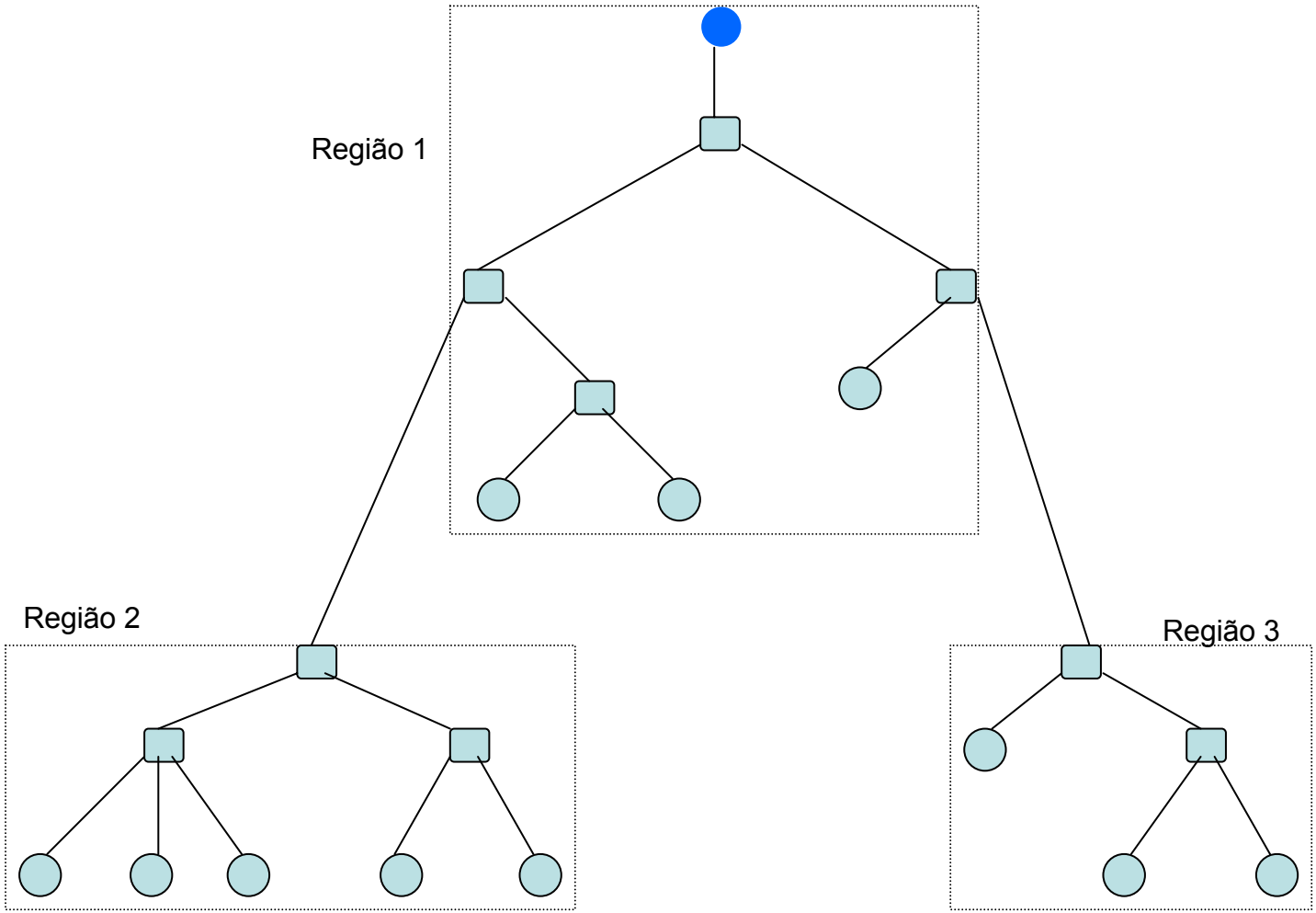
Receptor

Servidor de reparação

Randomized Reliable Multicast Protocol (RRMP): introdução

- Agrupa os receptores numa hierarquia, similiar aos protocolos baseados em árvore
- A responsabilidade da recuperação de erros é aleatoriamente distribuída entre todos os membros do grupo
- Elimina implosão de mensagens e proporciona boa recuperação de erros local
- Quando um receptor detecta a perda de uma mensagem envia um pedido aleatóriamente a um dos receptores na sua região
- A fiabilidade de protocolo não é deterministica, mas sim probabilista

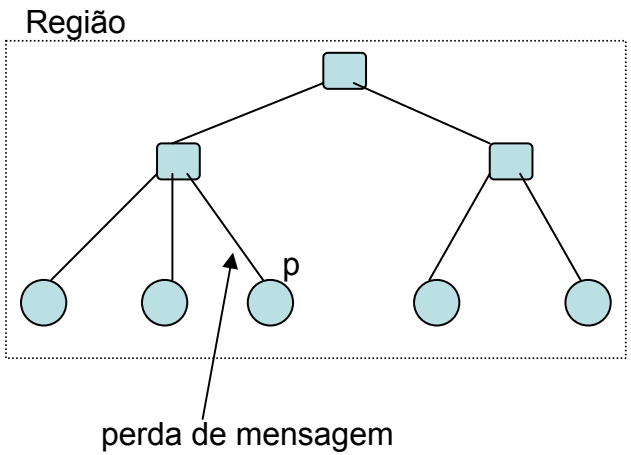
RRMP: regiões locais numa estrutura hierarquica



RRMP: reparação de erros

- Existem dois tipos de perdas de mensagens
  - Local: a perda afecta apenas uma porção de receptores de uma região
  - Regional: a perda afecta todos os receptores de uma região
- O algoritmo de reparação de erros consiste em duas fases executadas paralelamente:
  - Fase de reparação local
  - Fase de reparação remota

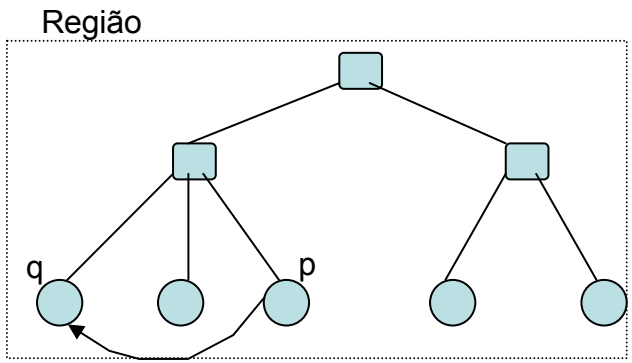
RRMP: reparação local



- Algoritmo
  - Processo *p* detecta a perda de uma mensagem *m*
  - Escolhe aleatoriamente um processo *q* na sua região e envia um pedido a *q*
  - *p* inicializa um contador baseado no tempo de ida-e-volta estimado para *q*
  - Ao receber o pedido, se estiver em posse de *m, q* envia-a a *p*, caso contrário ignora o pedido
  - Se o contador expirar antes de *p* receber *m*, voltar ao passo 2

RRMP: reparação local

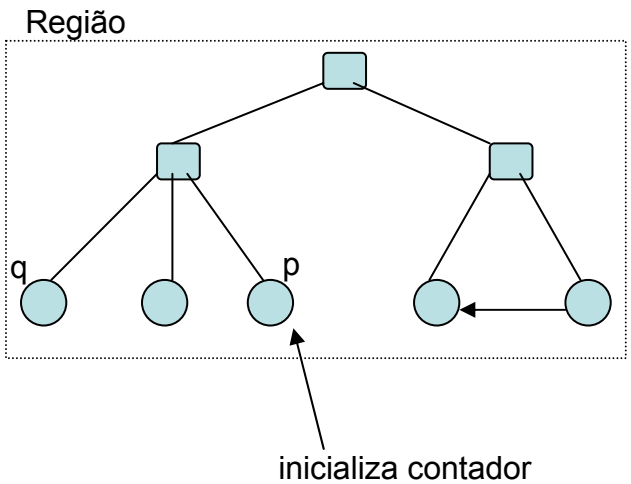
- Algoritmo
  - Processo *p* detecta a perda de uma mensagem *m*
  - Escolhe aleatoriamente um processo *q* na sua região e envia um pedido a *q*
  - *p* inicializa um contador baseado no tempo de ida-e-volta estimado para *q*
  - Ao receber o pedido, se estiver em posse de *m, q* envia-a a *p*, caso contrário ignora o pedido
  - Se o contador expirar antes de *p* receber *m*, voltar ao passo 2



pedidos locais

RRMP: reparação local

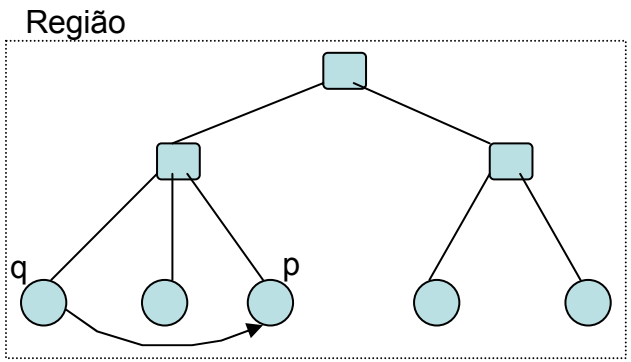
- Algoritmo
  - Processo *p* detecta a perda de uma mensagem *m*
  - Escolhe aleatoriamente um processo *q* na sua região e envia um pedido a *q*
  - *p* inicializa um contador baseado no tempo de ida-e-volta estimado para *q*
  - Ao receber o pedido, se estiver em posse de *m, q* envia-a a *p*, caso contrário ignora o pedido
  - Se o contador expirar antes de *p* receber *m*, voltar ao passo 2



pedidos locais

RRMP: reparação local

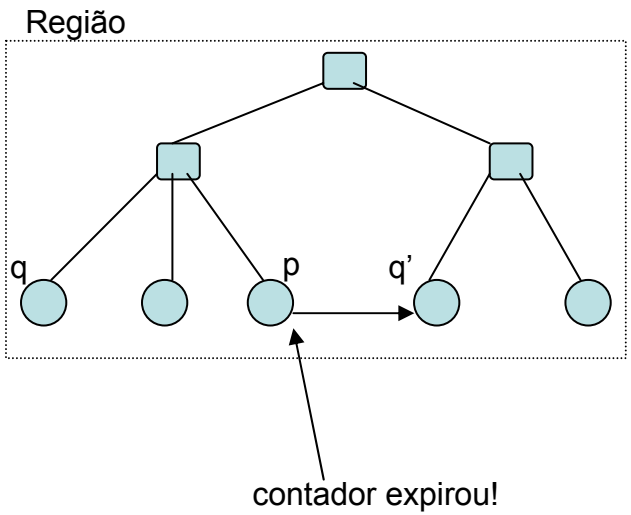
- Algoritmo
  - Processo *p* detecta a perda de uma mensagem *m*
  - Escolhe aleatoriamente um processo *q* na sua região e envia um pedido a *q*
  - *p* inicializa um contador baseado no tempo de ida-e-volta estimado para *q*
  - Ao receber o pedido, se estiver em posse de *m*, *q* envia-a a *p*, caso contrário ignora o pedido
  - Se o contador expirar antes de *p* receber *m*, voltar ao passo 2



pedidos locais

RRMP: reparação local

- Algoritmo
  - Processo *p* detecta a perda de uma mensagem *m*
  - Escolhe aleatoriamente um processo *q* na sua região e envia um pedido a *q*
  - *p* inicializa um contador baseado no tempo de ida-e-volta estimado para *q*
  - Ao receber o pedido, se estiver em posse de *m, q* envia-a a *p*, caso contrário ignora o pedido
  - Se o contador expirar antes de *p* receber *m*, voltar ao passo 2



pedidos locais

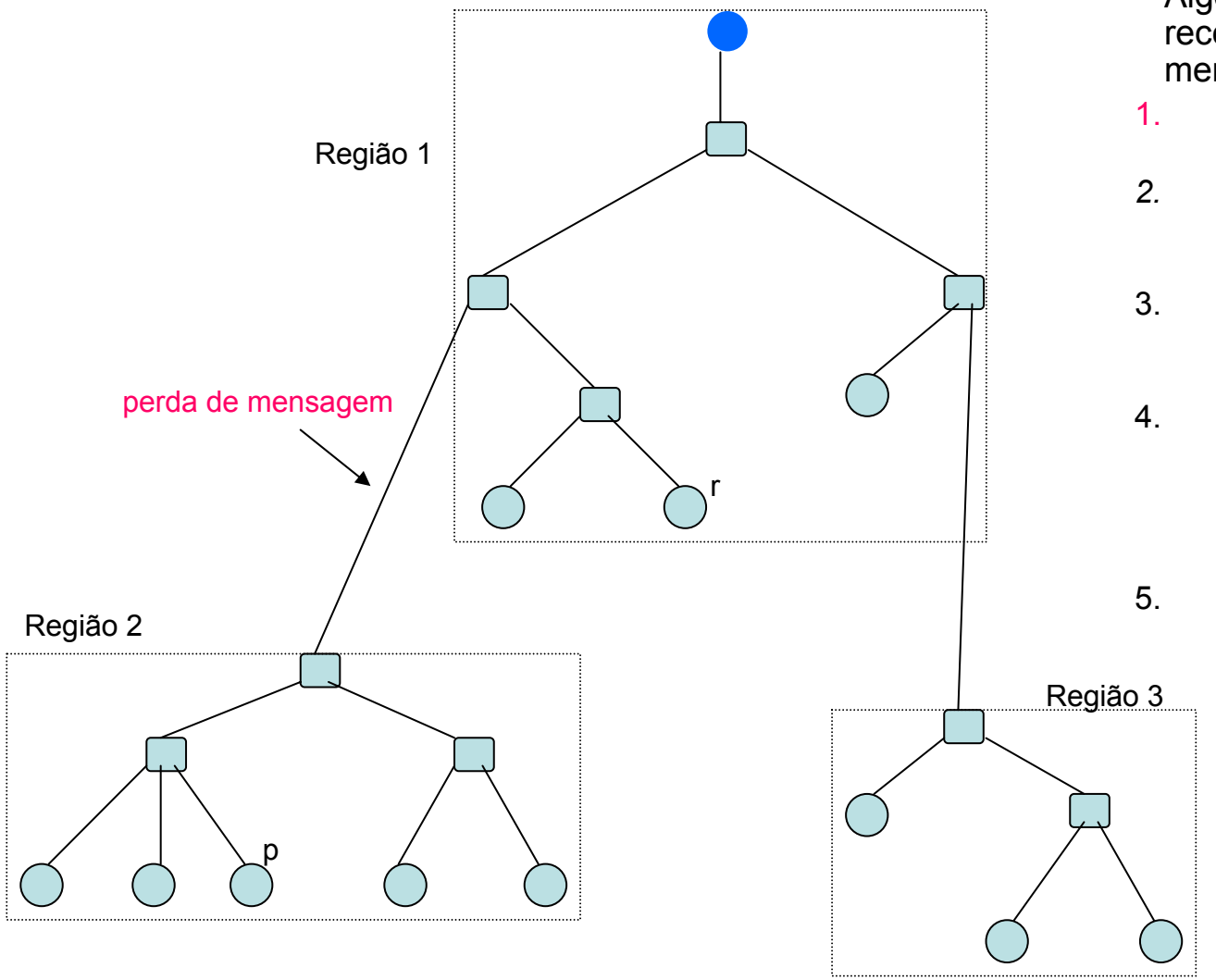
RRMP: reparação remota

- Se uma região inteira perdeu uma mensagem, a perda não pode ser reparada localmente
- Quando um receptor *p* detecta a perda da mensagem, escolhe aleatoriamente um receptor remoto *r* na sua região-pai e com uma probabilidade P, envia um pedido a *r*
- P é escolhida de forma a que o número de pedidos remotos enviado por todos os elementos da região seja uma constante λ
- Por exemplo, seja *n* o número de receptores numa região, Se P=1/n, então, em média, é enviado um pedido de reparação remoto é enviado (logo λ=1)

RRMP: reparação remota

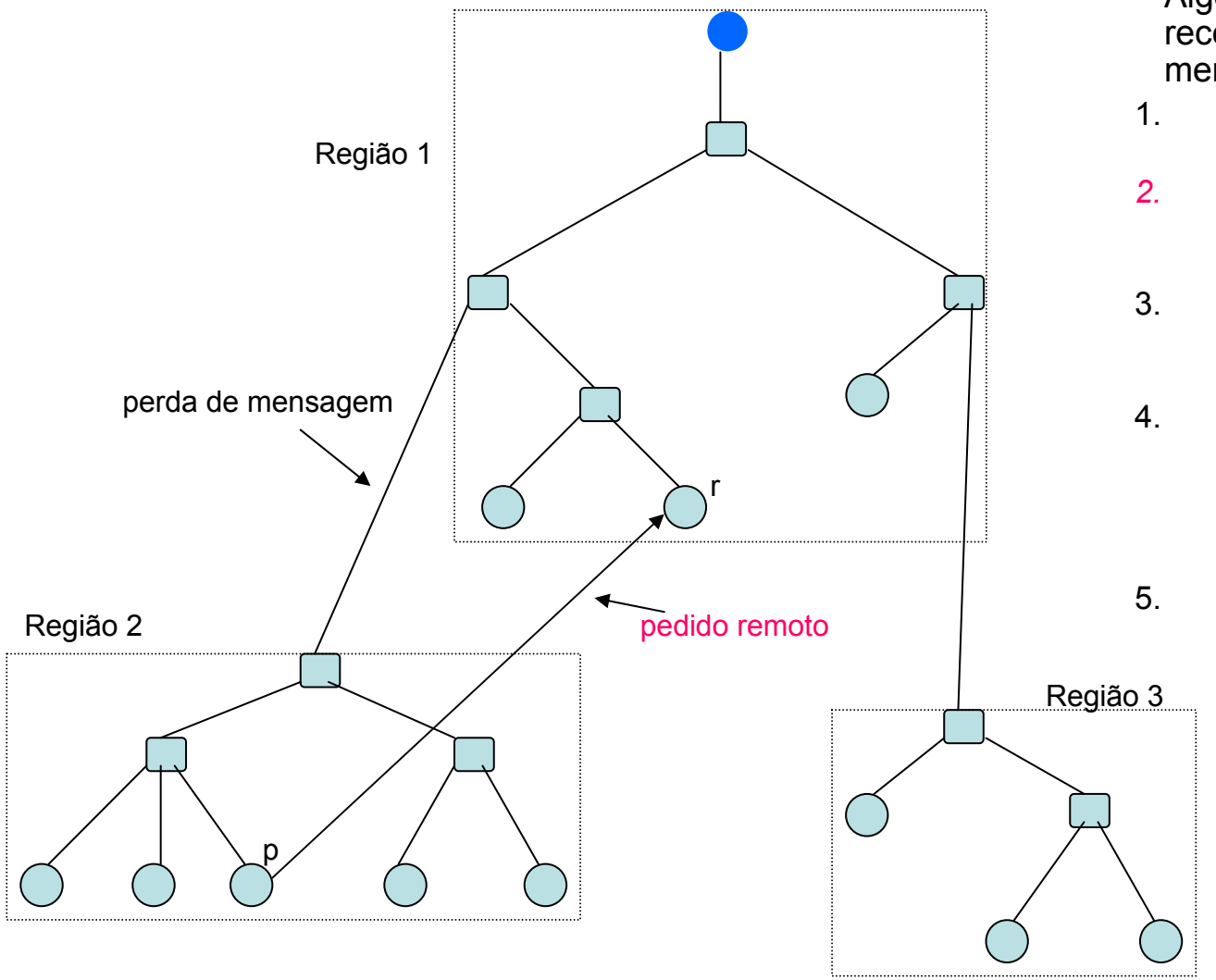
- Algoritmo (executado por todos os receptores na região que perdeu a mensagem)
  1. Processo *p* detecta a perda de uma mensagem *m*
  2. *p* envia com probabilidade P um pedido a um receptor remoto *r* escolhido aleatoriamente
  3. Inicia (sempre) um contador baseado no tempo de ida-e-volta entre *p* e *r*
  4. Se *p* não recebeu *m* antes do contador expirar volta ao passo 2
  5. Ao receber *m* de *r, p* difunde *m* para a sua região de modo a que todos os membros que partilhem a perda possam receber a mensagem

RRMP: reparação remota



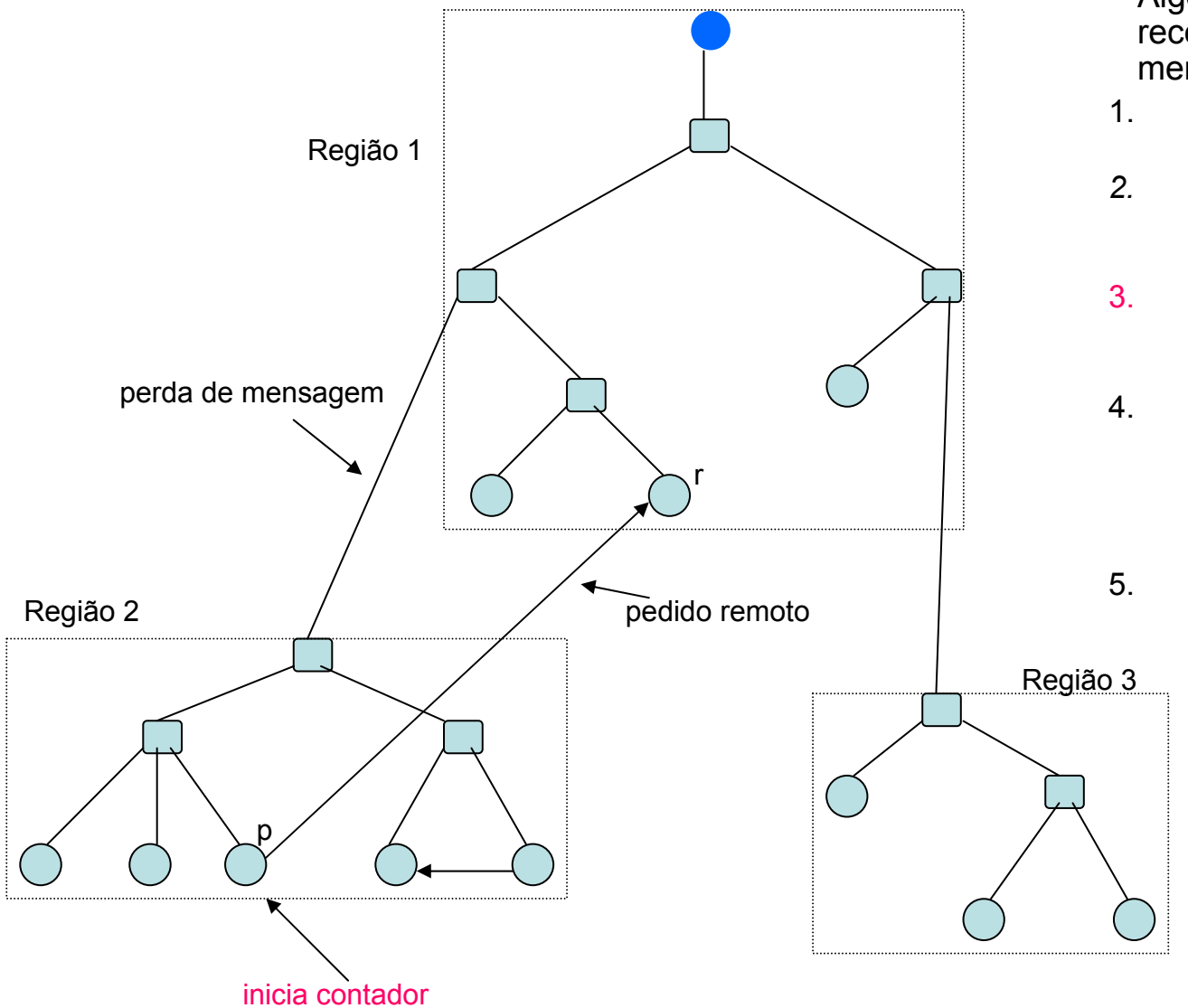
- Algoritmo (executado por todos os receptores na região que perdeu a mensagem)
  1. Processo *p* detecta a perda de uma mensagem *m*
  2. *p* envia com probabilidade P um pedido a um receptor remoto *r* escolhido aleatoriamente
  3. Inicia (sempre) um contador baseado no tempo de ida-e-volta entre *p* e *r*
  4. Ao receber *m* de *r, p* difunde *m* para a sua região de modo a que todos os membros que partilhem a perda possam receber a mensagem
  5. Se *p* não recebeu *m* antes do contador expirar volta ao passo 2

RRMP: reparação remota



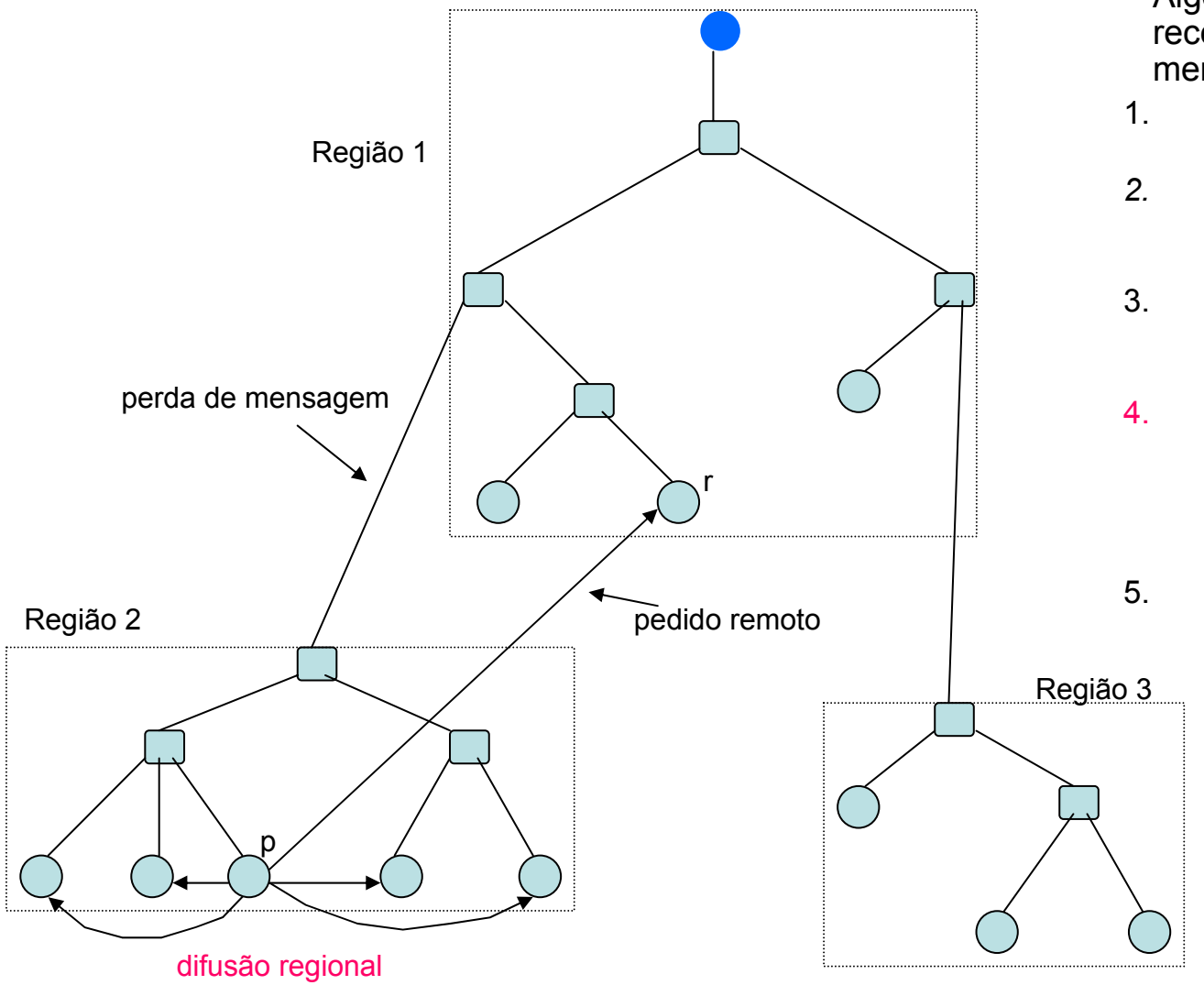
- Algoritmo (executado por todos os receptores na região que perdeu a mensagem)
  1. Processo *p* detecta a perda de uma mensagem *m*
  2. *p* envia com probabilidade P um pedido a um receptor remoto *r* escolhido aleatoriamente
  3. Inicia (sempre) um contador baseado no tempo de ida-e-volta entre *p* e *r*
  4. Ao receber *m* de *r*, *p* difunde *m* para a sua região de modo a que todos os membros que partilhem a perda possam receber a mensagem
  5. Se *p* não recebeu *m* antes do contador expirar volta ao passo 2

RRMP: reparação remota



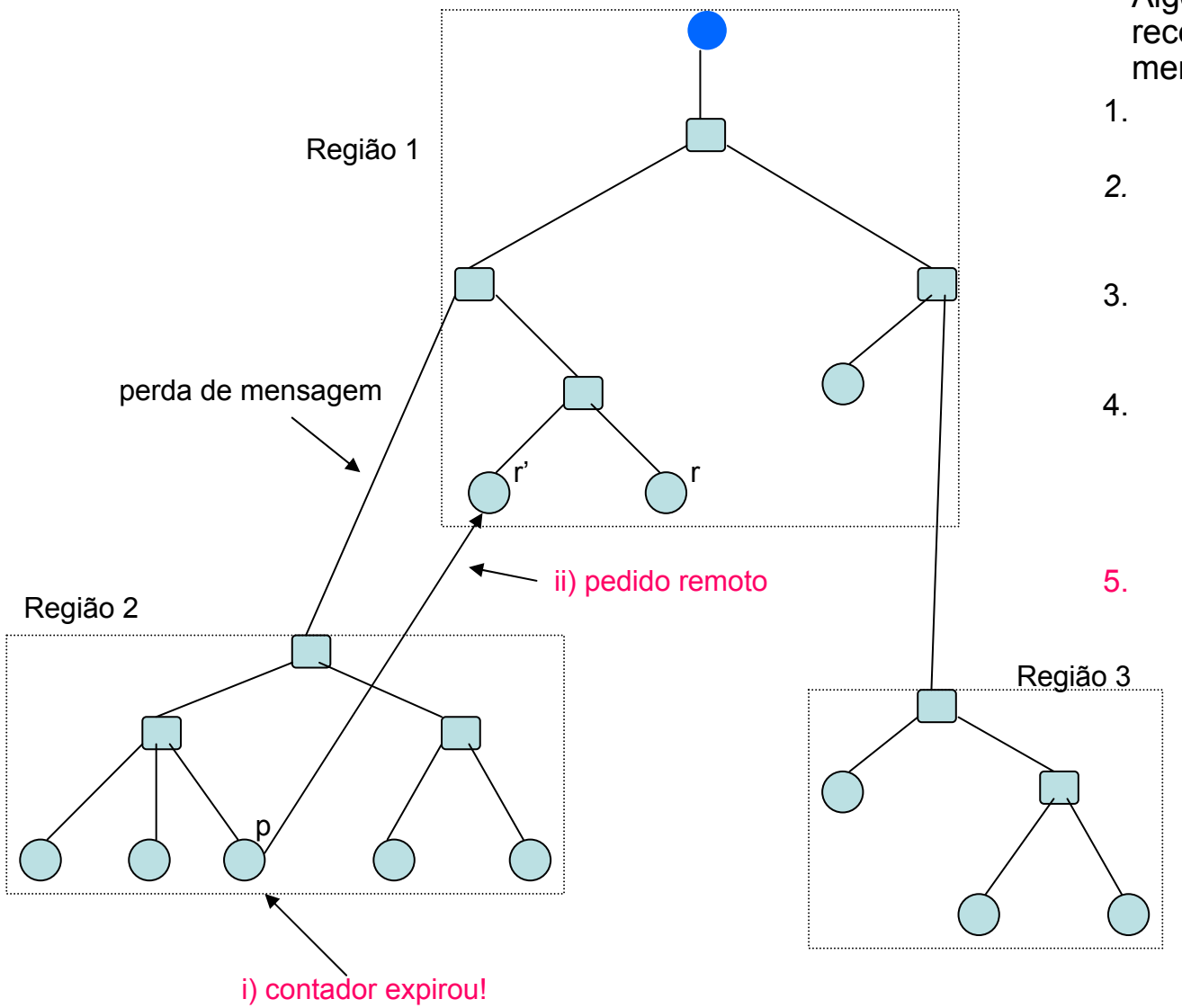
- Algoritmo (executado por todos os receptores na região que perdeu a mensagem)
  1. Processo *p* detecta a perda de uma mensagem *m*
  2. *p* envia com probabilidade P um pedido a um receptor remoto *r* escolhido aleatoriamente
  3. Inicia (sempre) um contador baseado no tempo de ida-e-volta entre *p* e *r*
  4. Ao receber *m* de *r*, *p* difunde *m* para a sua região de modo a que todos os membros que partilhem a perda possam receber a mensagem
  5. Se *p* não recebeu *m* antes do contador expirar volta ao passo 2

RRMP: reparação remota



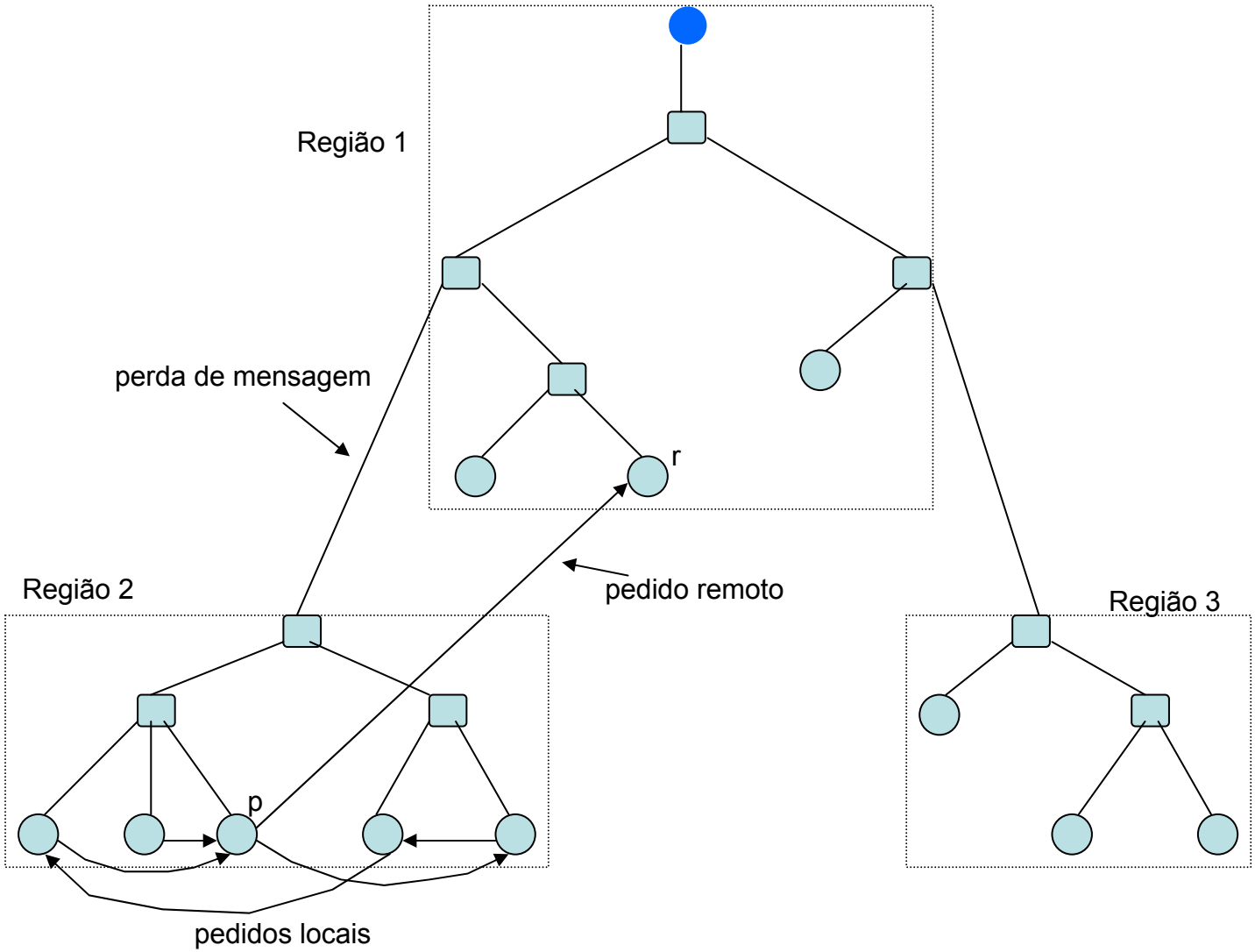
- Algoritmo (executado por todos os receptores na região que perdeu a mensagem)
  1. Processo *p* detecta a perda de uma mensagem *m*
  2. *p* envia com probabilidade P um pedido a um receptor remoto *r* escolhido aleatoriamente
  3. Inicia (sempre) um contador baseado no tempo de ida-e-volta entre *p* e *r*
  4. Ao receber *m* de *r*, *p* difunde *m* para a sua região de modo a que todos os membros que partilhem a perda possam receber a mensagem
  5. Se *p* não recebeu *m* antes do contador expirar volta ao passo 2

RRMP: reparação remota



- Algoritmo (executado por todos os receptores na região que perdeu a mensagem)
  1. Processo *p* detecta a perda de uma mensagem *m*
  2. *p* envia com probabilidade P um pedido a um receptor remoto *r* escolhido aleatoriamente
  3. Inicia (sempre) um contador baseado no tempo de ida-e-volta entre *p* e *r*
  4. Ao receber *m* de *r, p* difunde *m* para a sua região de modo a que todos os membros que partilhem a perda possam receber a mensagem
  5. Se *p* não recebeu *m* antes do contador expirar volta ao passo 2

RRMP: reparação remota e local



RRMP: medição dos tempos de ida-e-volta

- O algoritmo necessita que um receptor tenha noção dos tempos de ida-e-volta para os seus vizinhos locais como para os receptores na sua região-pai

- Estes tempos de ida-e-volta são usados para calcular os tempos de “timeout” no envio dos pedidos

RRMP: medição dos tempos de ida-e-volta

- Medição dos tempos de ida-e-volta para os membros locais
  - Quando um receptor *p* envia um pedido a outro membro local *q*, anexa uma estampilha temporal ao pedido
  - Esta estampilha temporal é depois copiada na resposta de *q* (assumindo que *q* está em posse da mensagem pedida)
  - O tempo de ida-e-volta é medido por *p* subtraindo o tempo de recepção da resposta ao tempo indicado pela estampilha temporal

RRMP: medição dos tempos de ida-e-volta

- Medição dos tempos de ida-e-volta para os membros remotos
  - É mais complicado porque um receptor remoto *r,* não envia necessariamente a resposta imediatamente à recepção de um pedido
  - O truque é fazer com que *r* inclua na resposta o intervalo tempo Δ que levou entre receber o pedido e enviar a resposta
  - Para além disso, quando *p* recebe uma resposta de *r*, inclui a sua estimativa do tempo de ida-e-volta ao difundir a mensagem de recuperação para a sua região

RRMP: medição dos tempos de ida-e-volta

- Medição dos tempos de ida-e-volta para os membros remotos (cont.)
  - Para evitar que a estimativa dos tempos de ida-e-volta se ressinta em periodos de pouca perda de mensagens, cada receptor define um intervalo máximo de tempo em que se não existirem pedidos, envia um pedido especial RTT_QUERY que despoleta uma resposta imediata RTT_RESPONSE

RRMP: formação da hierarquia

- 1) Estabelecimento das regiões locais
  - As regiões são criadas com base em domínios administrativos
  - Membros de uma região trocam periodicamente mensagens de sessão locais para tomarem conhecimento dos seus vizinhos e estimarem a dimensão da região
  - Uma mensagem de sessão local transmitida por *p* contém o maior número de sequência que *p* recebeu até ao momento do emissor

RRMP: formação da hierarquia

- 2) Estabelecimento da hierarquia de regiões
  - A ideia básica é cada membro reunir informação sobre os processos de modo a poder estabelecer uma região-pai
  - Cada membro anuncia periodicamente a sua presença difundindo uma mensagem global de sessão
  - Para reduzir o consumo de banda cada membro só difunde uma mensagem global de sessão num intervalo de tempo Tgs com probabilidade λ'/n

RRMP: formação da hierarquia

- 2) Estabelecimento da hierarquia de regiões (cont.)
  - Quando um membro *p* recebe uma mensagem global de sessão de um membro remoto *r* tem que decidir se selecciona *r* como membro da sua região-pai
  - Membros da mesma região que o emissor não têm região-pai
  - A informação necessária para o estabelecimento da hierarquia está toda contida nas mensagens globais de sessão (número de hops desde o emissor)

RRMP: formação da hierarquia

- 2) Estabelecimento da hierarquia de regiões (cont.)
  - Algoritmo
    - 1) *p* verifica se *r* é um membro “upstream”: se *r* está mais perto do emissor que *p* e *p* está mais perto de *r* que do emissor
    - 2) Se *r* é um membro “upstream”, então *p* compara a sua distância para *r* com a de outros membros “upstream” dos quais recebeu mensagens recentemente e selecciona um conjunto dos que estão mais perto como membros da sua região-pai

RRMP: formação da hierarquia

- 2) Estabelecimento da hierarquia de regiões (cont.)
  - *p* mantém uma lista dos seus membros “upstream” que compõem a região-pai
  - Para cada elemento *r* contém: endereço, número de hops desde *r*, e um contador
  - O contador é posto a zero cada vez que *p* recebe uma mensagem de *r*, quando expira *r* é removido da lista
  - Propriedade da lista: a distância desde *p* até ao membro mais longínquo na lista não pode exceder H hops desde o membro mais próximo na lista

RRMP: simulação e resultados

- Foi efectuada uma avaliação de performance do RRMP utilizando o simulador ns2
- Como comparação é usado o protocolo TRMP, que é um protocolo baseado em árvore com reparação de erros através de servidores de recuperação
- Para o TRMP cada região tem um servidor de reparação colocado da raíz da região

RRMP: simulação e resultados

- Para o RRMP, cada simulação é iniciada com um período de “bootstrap” de 30 segs em que o emissor difunde mensagens globais de sessão a cada segundo para que cada receptor consiga contar o número de hops desde o emissor
- Após este periodo o emissor envia mensagens de 1Kbyte a uma velocidade constante de 50 msg/s durante 10 mins
- Um total de 30.000 mensagens são recebidas por cada receptor e são entregues à aplicação por ordem FIFO
- Paralelamente, é gerado tráfico em “background” na rede de modo a ‘forçar’ perdas de mensagens

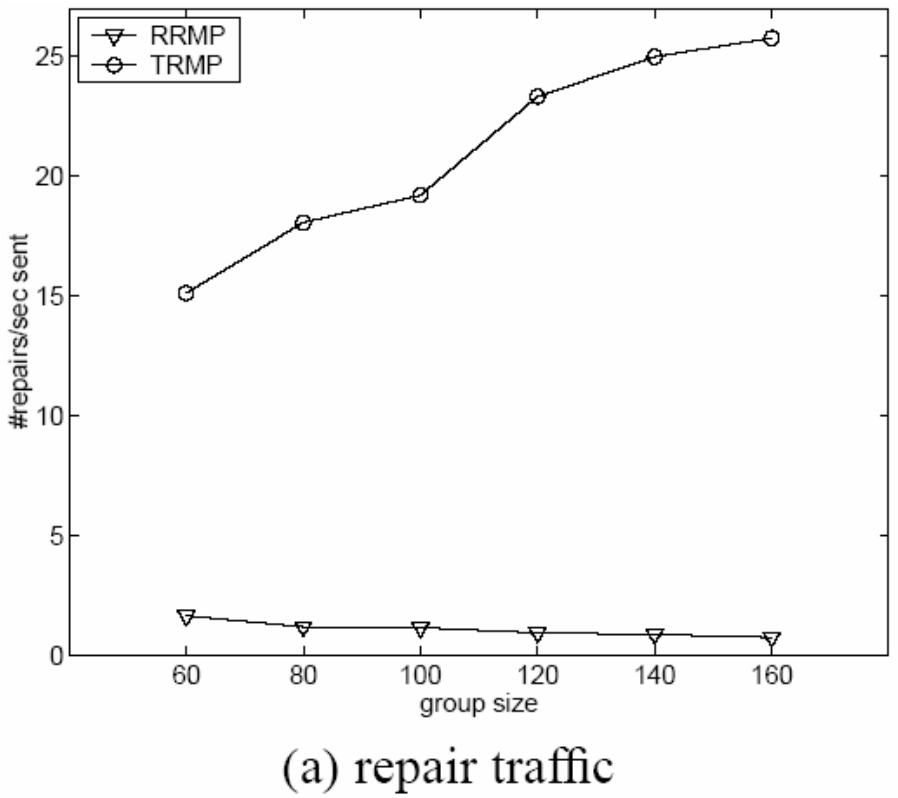
RRMP: simulação e resultados

- Parâmetros de configuração para o RRMP

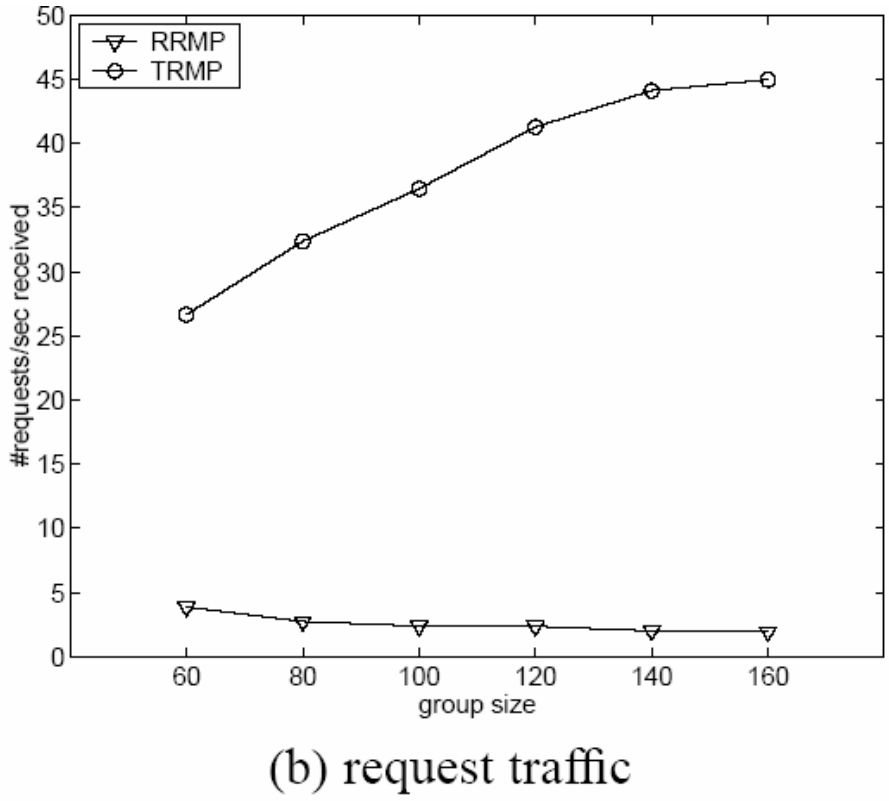
T_{maxl}	1 second
T_{maxr}	5 second
T_{gs}	1 second
T_{ls}	1 second
H	4 hops
λ'	2

RRMP: simulação e resultados

- Balanceamento de carga
  - Comparação de tráfego de *(a)* *reparação* e *(b)* *pedidos* à medida que a dimensão do grupo aumenta com λ=4



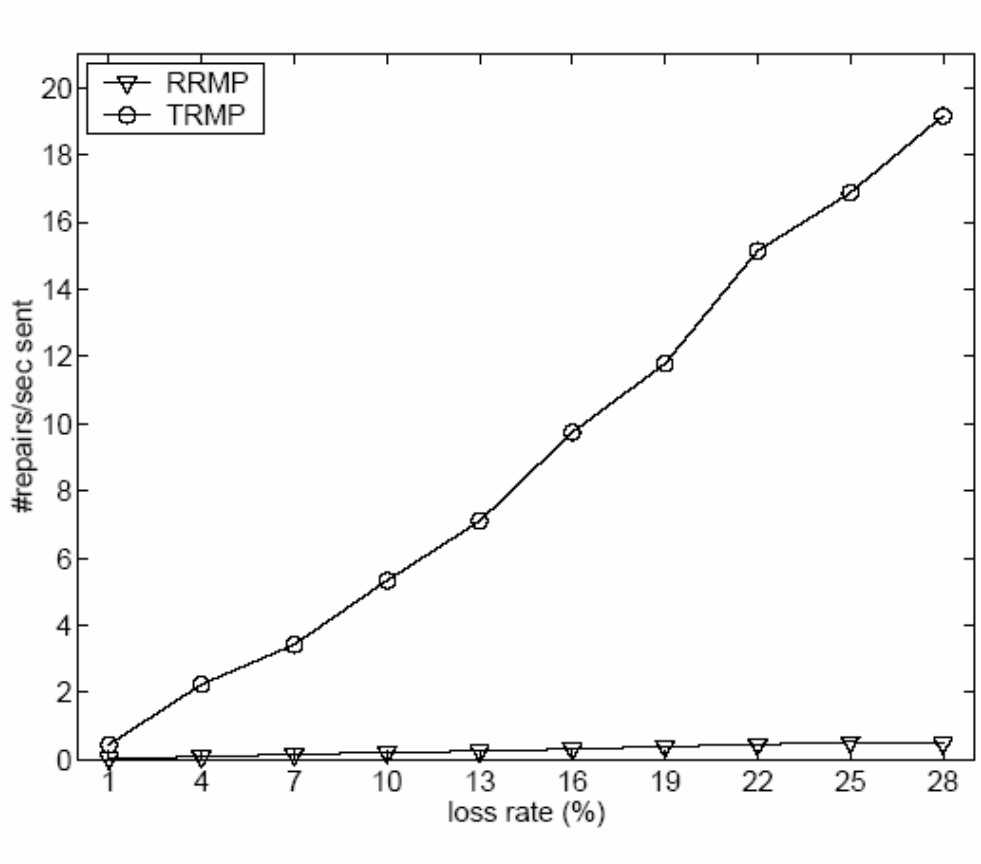
(a) repair traffic



(b) request traffic

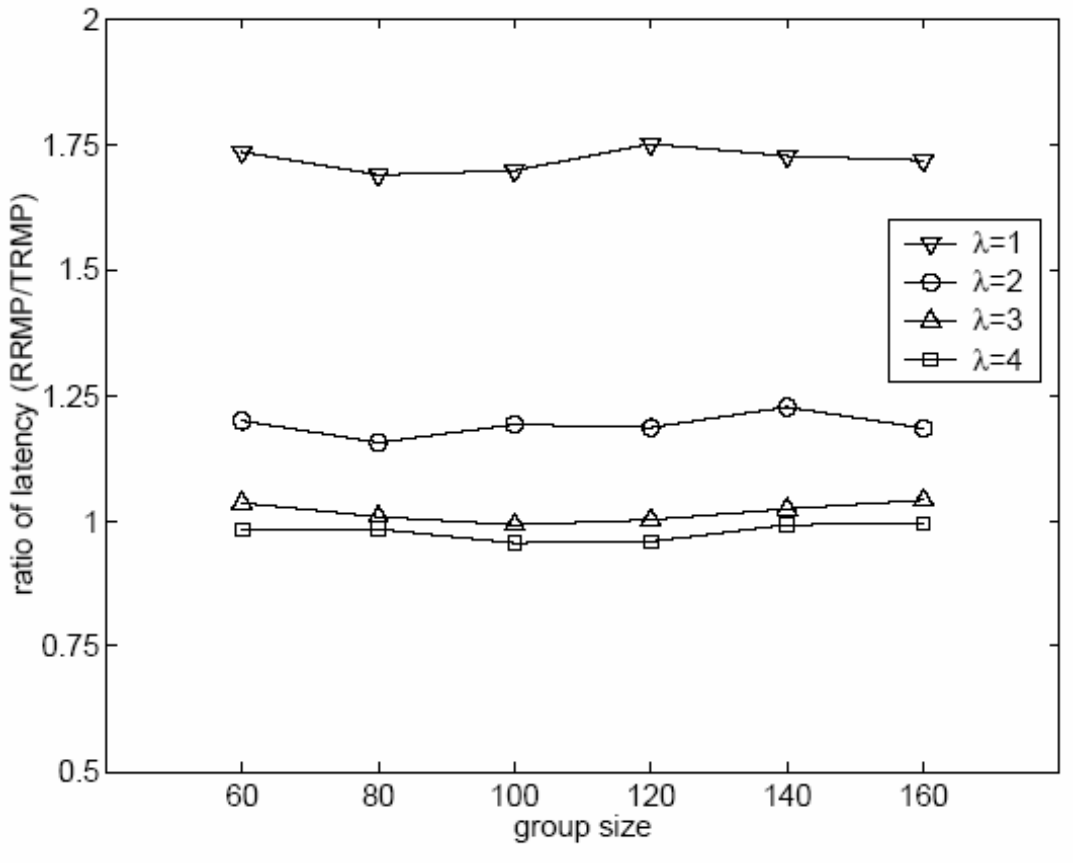
RRMP: simulação e resultados

- Balanceamento de carga
  - Comparação do tráfego de reparação para um receptor que perde mensagens (grupo com 160 elementos)



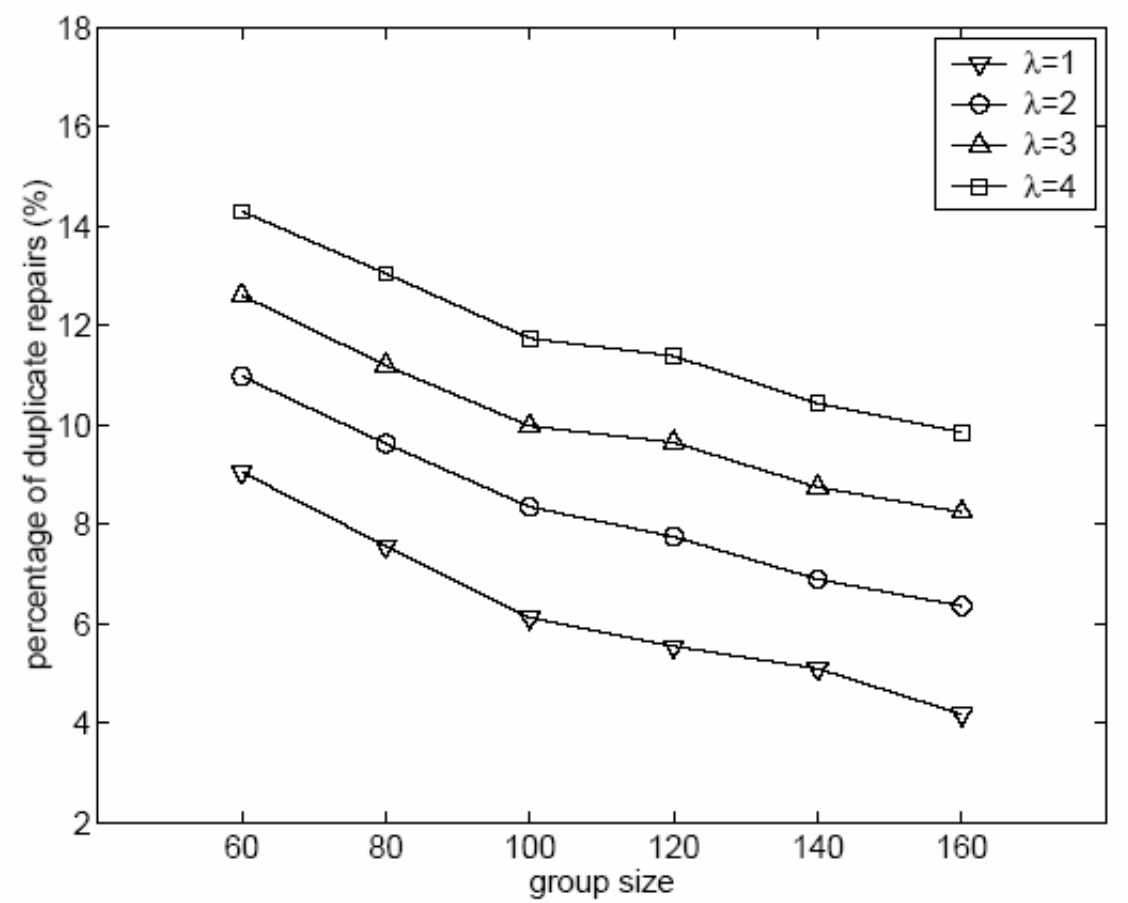
RRMP: simulação e resultados

- Latencia de recuperação
  - Latencia na recuperação de erros no RRMP



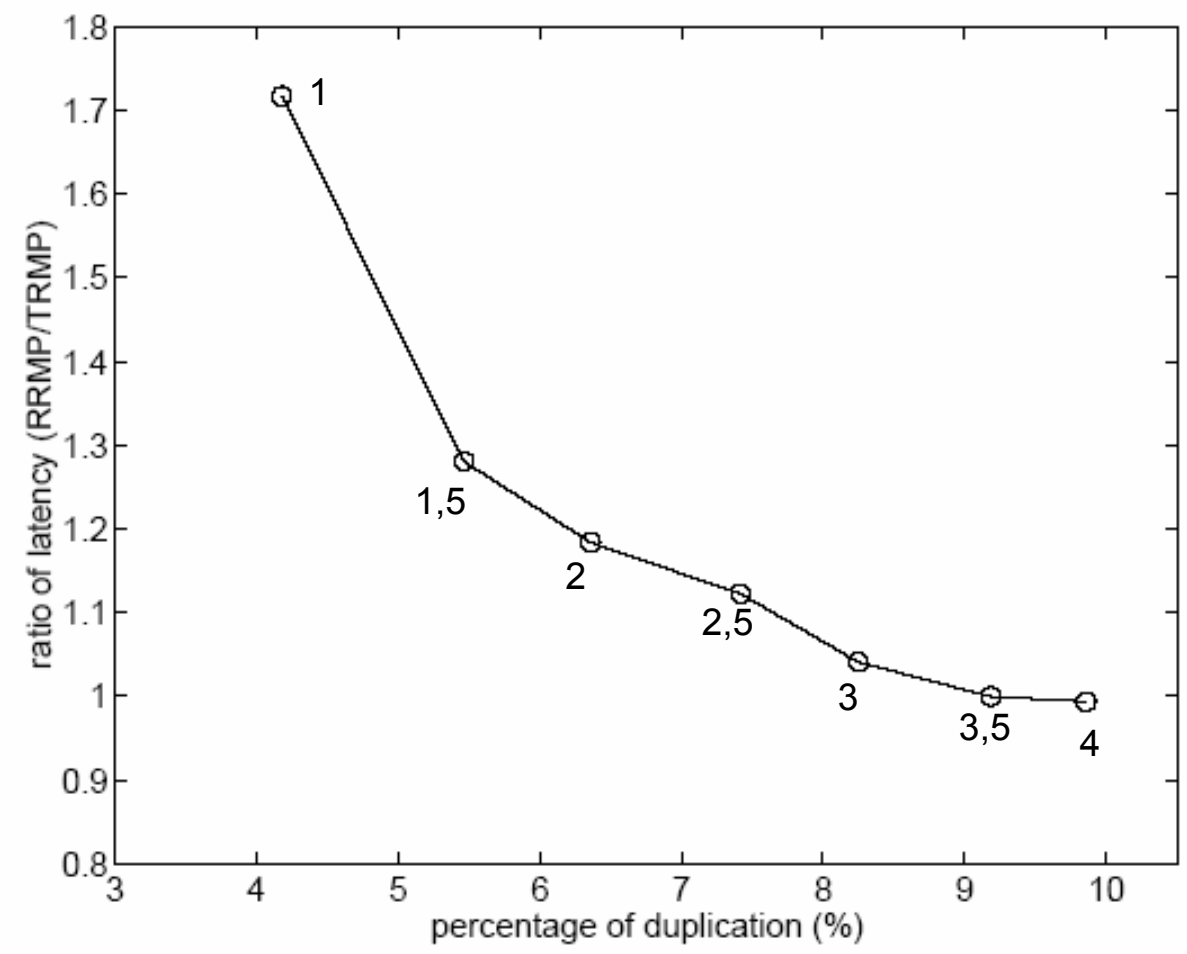
RRMP: simulação e resultados

- Duplicação de reparações



RRMP: simulação e resultados

- “tradeoff” entre latencia de recuperação e duplicação de reparações para diferentes valores de λ



RRMP: conclusões

- Quando comparado com os protocolos baseados em árvore tradicionais, RRMP consegue um melhor balanceamento de carga distribuindo a responsabilidade de reparação de erros por todos os membros do grupo
- A robustez contra a falha de processos também é melhorada, não existem pontos únicos de falha
- A latência na recuperação de erros é melhorada através da execução paralela das fases de recuperação local e remota
- Os parâmetros de configuração podem ser ajustados consoante as necessidades da aplicação, tornando o algoritmo flexível

Referências

- Artigo original: A Randomized Error Recovery Algorithm for Reliable Multicast, Zhen Xiao, Kenneth P. Birman

- IP Multicast: wikipedia http://en.wikipedia.org/wiki/Multicast

- Simulador de rede ns2: http://www.isi.edu/nsnam/ns