

# ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos

## Aula 10 – Comunicação Orientada a Mensagens e *Multicast*

Norton Trevisan Roman

26 de maio de 2022

# Comunicação Orientada a Mensagens

- Comunicação Transiente
- Comunicação Persistente
- Comunicação *Multicast*

# Comunicação Orientada a Mensagens

- **Comunicação Transiente**
- Comunicação Persistente
- Comunicação *Multicast*

## Sockets

- Modelo simples orientado a mensagens oferecido pela camada de transporte

## Sockets

- Modelo simples orientado a mensagens oferecido pela camada de transporte
- Conceitualmente, um ponto de comunicação
  - Ao qual uma aplicação pode escrever dados a serem enviados pela rede
  - Do qual dados que chegam podem ser lidos

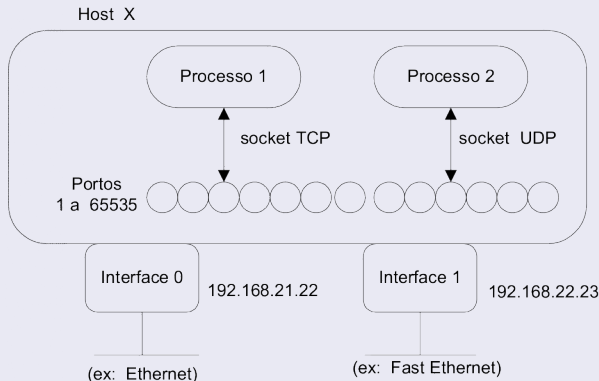
## Sockets

- Modelo simples orientado a mensagens oferecido pela camada de transporte
- Conceitualmente, um ponto de comunicação
  - Ao qual uma aplicação pode escrever dados a serem enviados pela rede
  - Do qual dados que chegam podem ser lidos
- Forma uma abstração sobre a porta real usada pelo SO

# Comunicação Transiente

## Sockets

- Em unix, sockets são um tipo de arquivo que conecta processos a suas respectivas portas



# Comunicação Transiente

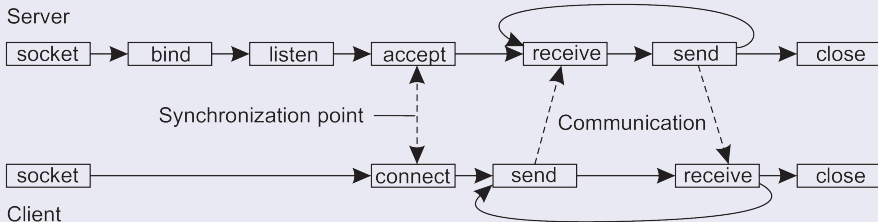
## Berkeley socket interface (POSIX)

SOCKET	Cria um novo ponto de comunicação
BIND	Especifica um endereço local (IP + porta) ao socket
LISTEN	Diz ao SO o número máximo de pedidos de conexão a serem recebidos
ACCEPT	Bloqueia o executor até receber um pedido de estabelecimento de conexão
CONNECT	Tenta estabelecer uma conexão
SEND	Envia dados pela conexão
RECEIVE	Recebe dados pela conexão
CLOSE	Libera a conexão



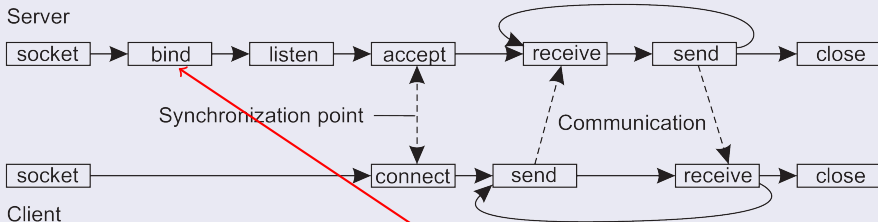
# Comunicação Transiente

## Sockets: Modelo de comunicação



# Comunicação Transiente

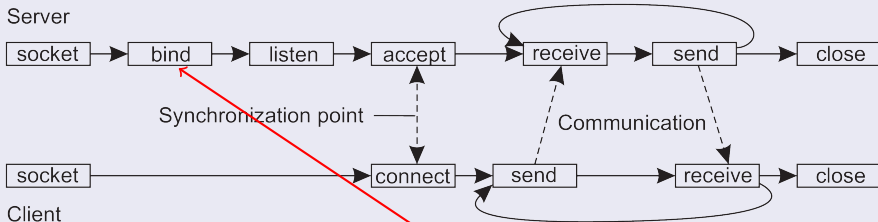
## Sockets: Modelo de comunicação



O servidor vincula o IP de sua máquina (juntamente com um número de porta) ao socket

# Comunicação Transiente

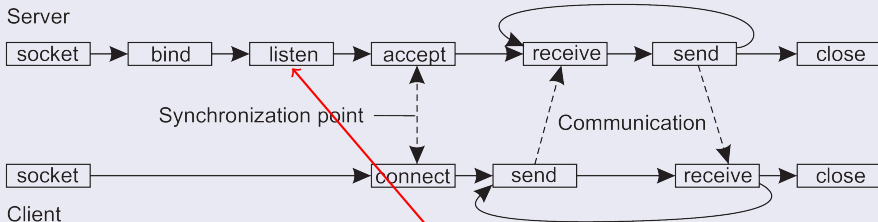
## Sockets: Modelo de comunicação



Isso diz ao SO que o servidor quer receber mensagens somente nesse endereço e porta específicos

# Comunicação Transiente

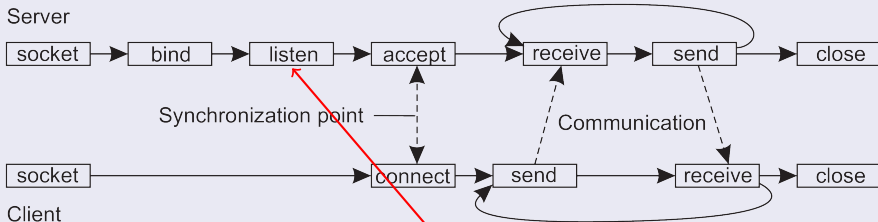
## Sockets: Modelo de comunicação



**listen é chamada somente em caso de comunicação orientada à conexão (necessita do estabelecimento da conexão antes do envio da mensagem)**

# Comunicação Transiente

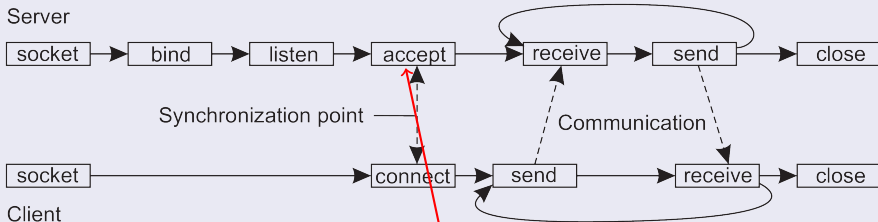
## Sockets: Modelo de comunicação



Chamada não bloqueante que faz com que o SO reserve *buffers* o suficiente para um número máximo de requisições de conexão pendentes

# Comunicação Transiente

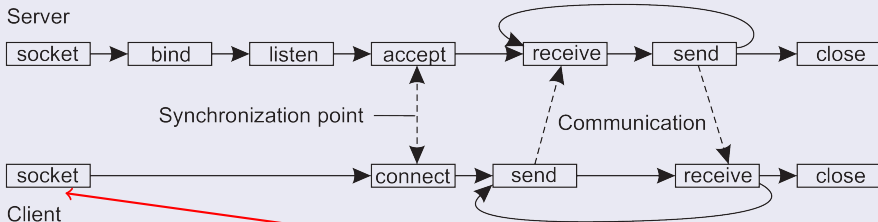
## Sockets: Modelo de comunicação



accept bloqueia quem a chama até que um pedido de conexão chegue

# Comunicação Transiente

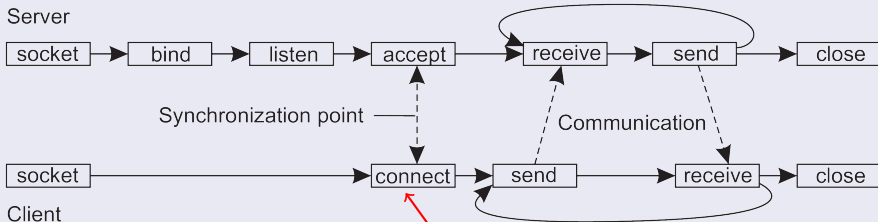
## Sockets: Modelo de comunicação



No cliente, o socket é criado, mas sem precisar vinculá-lo explicitamente a um endereço (o SO alocará uma porta dinamicamente)

# Comunicação Transiente

## Sockets: Modelo de comunicação

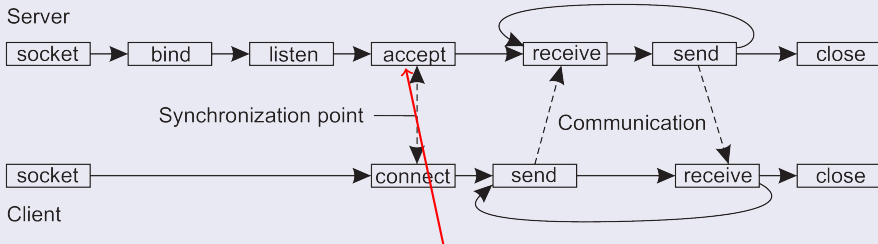


connect faz a requisição para conexão e bloqueia o cliente até que esta seja estabelecida



# Comunicação Transiente

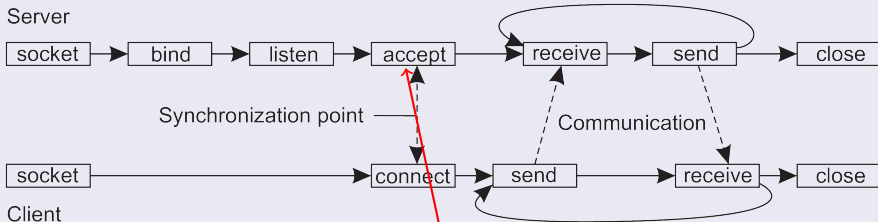
## Sockets: Modelo de comunicação



Quando a requisição chega, o SO cria um novo *socket* com as mesmas propriedades do original, e o retorna a quem chamou `accept`

# Comunicação Transiente

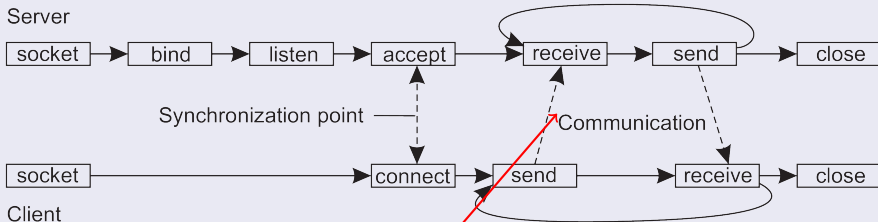
## Sockets: Modelo de comunicação



Isso permite que o servidor crie um novo processo para lidar com a comunicação por esta conexão

# Comunicação Transiente

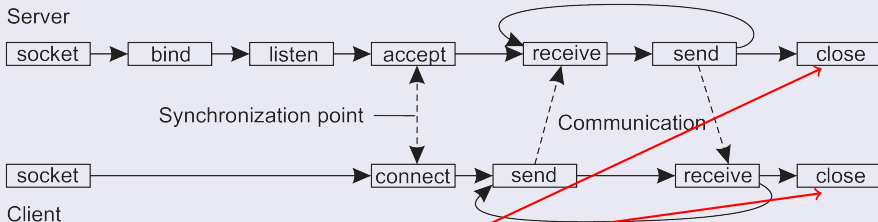
## Sockets: Modelo de comunicação



Começa o processo de rendez-vous

# Comunicação Transiente

## Sockets: Modelo de comunicação



Terminar a conexão é simétrica: é estabelecida quanto tanto o cliente quanto o servidor executam close

# Comunicação Transiente

## *Message-Passing Interface* (MPI)

- Por vezes, *sockets* não são suficientes
  - Suportam apenas operações `send` e `receive` simples

# Comunicação Transiente

## *Message-Passing Interface* (MPI)

- Por vezes, *sockets* não são suficientes
  - Suportam apenas operações `send` e `receive` simples
- Projetados para comunicação usando protocolos de uso geral, como TCP/IP
  - Não para protocolos proprietários de alta velocidade

## *Message-Passing Interface* (MPI)

- Por vezes, *sockets* não são suficientes
  - Suportam apenas operações *send* e *receive* simples
- Projetados para comunicação usando protocolos de uso geral, como TCP/IP
  - Não para protocolos proprietários de alta velocidade
- A necessidade de independência de hardware e plataforma levou à definição de um padrão para passagem de mensagem
  - *Message-Passing Interface* (MPI)

## *Message-Passing Interface* (MPI)

- Projetada para aplicações em paralelo, com comunicação transiente



## *Message-Passing Interface (MPI)*

- Projetada para aplicações em paralelo, com comunicação transiente
- Assume que a comunicação ocorre dentro de um grupo conhecido de processos
  - Cada grupo recebe um identificador
  - Cada processo dentro do grupo recebe um identificador local
  - O par (idGrupo, idProcesso) identifica unicamente a fonte ou destino de uma mensagem (e é usado em vez de um endereço)

# Comunicação Transiente


## MPI: Operações

<code>MPI_bsend</code>	Adiciona mensagem a buffer de envio local
<code>MPI_send</code>	Envia a mensagem e espera até ser copiada a buffer remoto ou local
<code>MPI_ssend</code>	Envia a mensagem e espera até a transmissão começar
<code>MPI_sendrecv</code>	Envia a mensagem e espera pela resposta
<code>MPI_issend</code>	Passa referência a mensagem de saída e continua
<code>MPI_issend</code>	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
<code>MPI_recv</code>	Recebe uma mensagem; bloqueia se não houver nenhuma
<code>MPI_irecv</code>	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Usada para  
comunicação  
transiente  
assíncrona



MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Usada para comunicação transiente assíncrona

O emissor submete a mensagem, que é copiada para um buffer no *runtime system* do MPI

MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Usada para comunicação transiente assíncrona

O emissor submete a mensagem, que é copiada para um buffer no *runtime system* do MPI

Após a mensagem ser copiada, o emissor continua

MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_issend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Versão síncrona  
de MPI\_bsend

MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Versão síncrona  
de MPI\_bsend

A depender da  
implementação,  
pode bloquear  
quem a  
chama até a  
mensagem ser  
copiada para  
o buffer local

MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Versão síncrona  
de MPI\_bsend

A depender da  
implementação,  
pode bloquear  
quem a  
chama até a  
mensagem ser  
copiada para  
o buffer local

Ou até o  
receptor iniciar  
uma operação  
de *receive*


MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia



# Comunicação Transiente

## MPI: Operações

Usada para  
comunicação  
síncrona




MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Usada para  
comunicação  
síncrona

O emissor  
bloqueia até  
a requisição  
ser aceita para  
processamento




MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Forma mais  
forte de  
comunicação  
síncrona




MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_issend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Forma mais forte de comunicação síncrona

O emissor envia uma requisição ao receptor e bloqueia até que este a responda



MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_issend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Forma mais forte de comunicação síncrona

O emissor envia uma requisição ao receptor e bloqueia até que este a responda

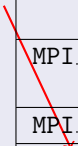
Corresponde ao RPC comum

MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_issend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

O emissor  
passa apenas  
um ponteiro  
para a mensa-  
gem ao *runtime*  
do MPI



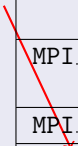
MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

O emissor  
passa apenas  
um ponteiro  
para a mensa-  
gem ao *runtime*  
do MPI

Continuando  
então a rodar  
normalmente

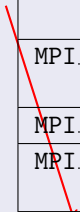


MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_issend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

O emissor  
passa apenas  
um ponteiro  
para a mensa-  
gem ao *runtime*  
do MPI



MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia



# Comunicação Transiente

## MPI: Operações

O emissor passa apenas um ponteiro para a mensagem ao *runtime* do MPI

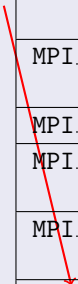
Quanto o *runtime* indicar que processou a mensagem, então o emissor saberá que o receptor aceitou a mensagem e está trabalhando nela

MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Chamado para  
receber uma  
mensagem



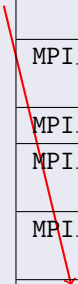
MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Chamado para  
receber uma  
mensagem

Bloqueia  
quem a chama  
até que a  
mensagem  
chegue

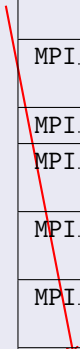


MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Transiente

## MPI: Operações

Variante  
assíncrona  
de MPI\_recv



MPI_bsend	Adiciona mensagem a buffer de envio local
MPI_send	Envia a mensagem e espera até ser copiada a buffer remoto ou local
MPI_ssend	Envia a mensagem e espera até a transmissão começar
MPI_sendrecv	Envia a mensagem e espera pela resposta
MPI_isend	Passa referência a mensagem de saída e continua
MPI_issend	Passa referência a mensagem de saída, e espera até que o receptor da referência comece
MPI_recv	Recebe uma mensagem; bloqueia se não houver nenhuma
MPI_irecv	Verifica se há mensagem recebida; não bloqueia

# Comunicação Orientada a Mensagens

- Comunicação Transiente
- **Comunicação Persistente**
- Comunicação *Multicast*

## *Middleware Orientado a Mensagens (MOM)*

- Também conhecidos como **Sistemas de Enfileiramento de Mensagens**

## *Middleware* Orientado a Mensagens (MOM)

- Também conhecidos como **Sistemas de Enfileiramento de Mensagens**
- Comunicação assíncrona e persistente graças ao uso de **filas** pelo *middleware*
  - Filas correspondem a *buffers* em servidores de comunicação

## *Middleware* Orientado a Mensagens (MOM)

- Também conhecidos como **Sistemas de Enfileiramento de Mensagens**
- Comunicação assíncrona e persistente graças ao uso de **filas** pelo *middleware*
  - Filas correspondem a *buffers* em servidores de comunicação
- Fornecem suporte a comunicação persistente assíncrona
  - Não exigem que emissor e receptor estejam ativos durante a transmissão da mensagem



# Comunicação Persistente

## MOM – Modelo de enfileiramento de mensagens

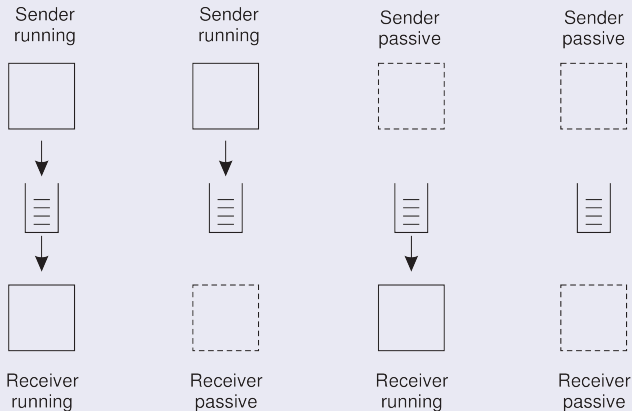
- As aplicações se comunicam pela inserção de mensagens em filas específicas
  - Ou uma fila privada por aplicação, para a qual outras aplicações podem enviar mensagens, ou fila compartilhada
  - O endereçamento é feito pela atribuição de nomes únicos às filas

## MOM – Modelo de enfileiramento de mensagens

- As aplicações se comunicam pela inserção de mensagens em filas específicas
  - Ou uma fila privada por aplicação, para a qual outras aplicações podem enviar mensagens, ou fila compartilhada
  - O endereçamento é feito pela atribuição de nomes únicos às filas
- A única garantia dada ao emissor é que a mensagem será inserida na fila do receptor
  - Nenhuma garantia é dada acerca de quanto ou se a mensagem será lida

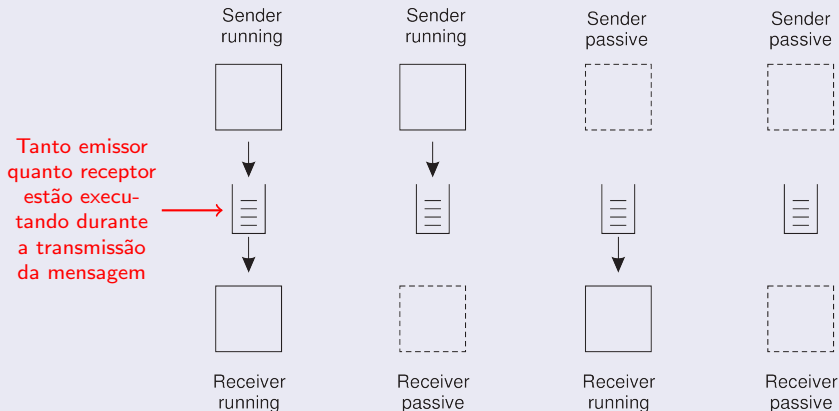
# Comunicação Persistente

## MOM – Modelos de execução emissor/receptor



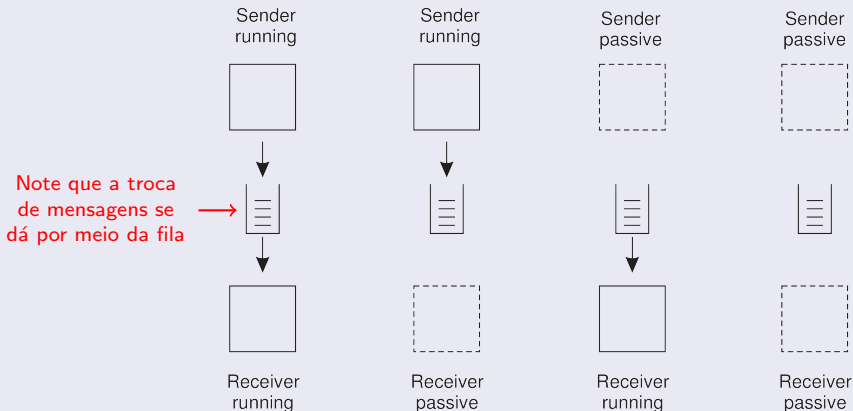
# Comunicação Persistente

## MOM – Modelos de execução emissor/receptor



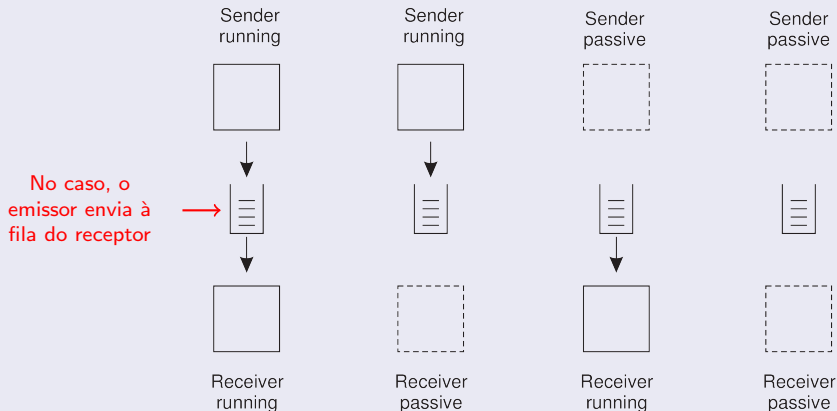
# Comunicação Persistente

## MOM – Modelos de execução emissor/receptor



# Comunicação Persistente

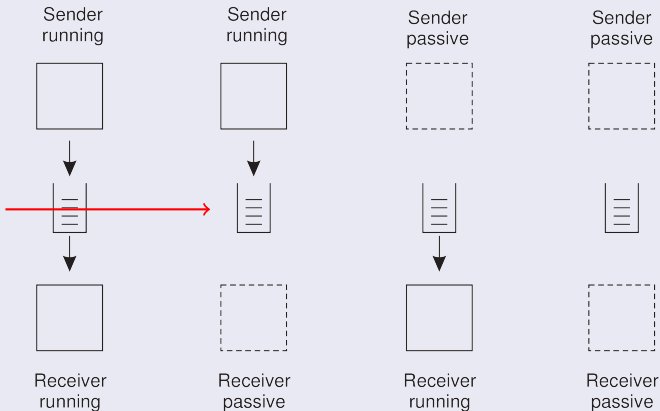
## MOM – Modelos de execução emissor/receptor



# Comunicação Persistente

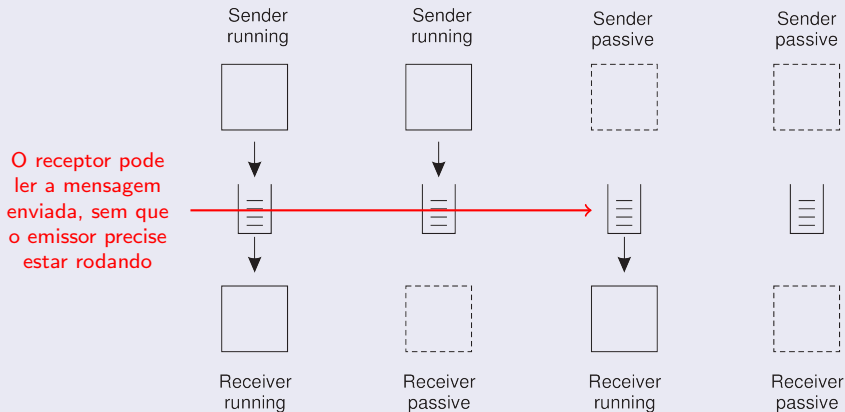
## MOM – Modelos de execução emissor/receptor

Somente o emissor está rodando, enquanto o receptor está em um estado em que a mensagem não pode ser entregue



# Comunicação Persistente

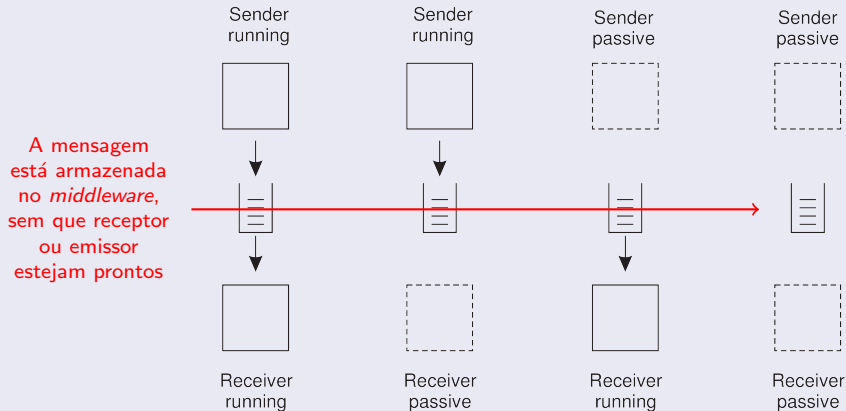
## MOM – Modelos de execução emissor/receptor





# Comunicação Persistente

## MOM – Modelos de execução emissor/receptor



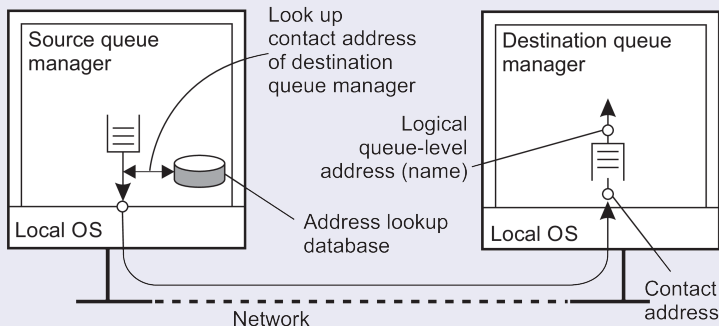
# Comunicação Persistente

## MOM – Interface básica

PUT	Adiciona uma mensagem à fila especificada; operação não bloqueante
GET	Bloqueia até que a fila especificada tenha alguma mensagem e remove a primeira mensagem
POLL	Verifica se a fila especificada tem alguma mensagem e remove a primeira. Nunca bloqueia
NOTIFY	Instala um tratador (função de <i>callback</i> ) para ser chamado sempre que uma mensagem for inserida em uma dada fila

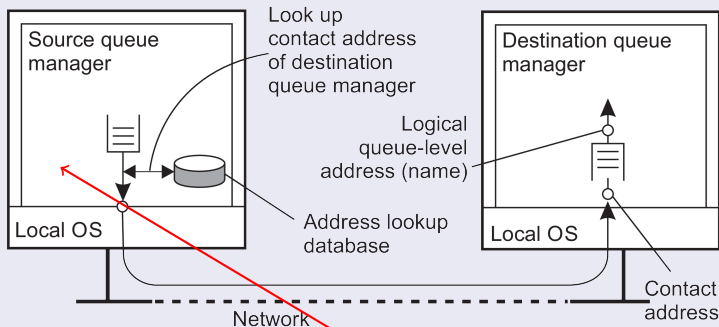
# Comunicação Persistente

## MOM – Arquitetura geral



# Comunicação Persistente

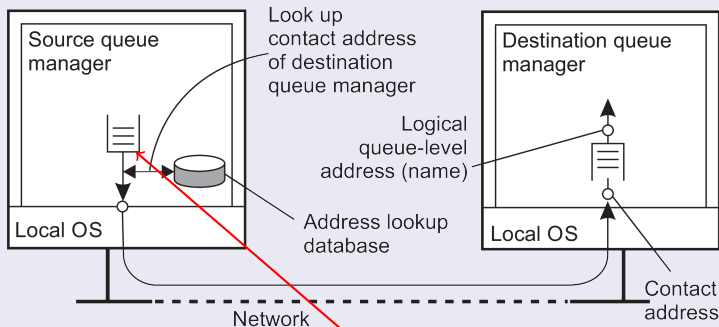
## MOM – Arquitetura geral



As filas são gerenciadas por gerenciadores de filas

# Comunicação Persistente

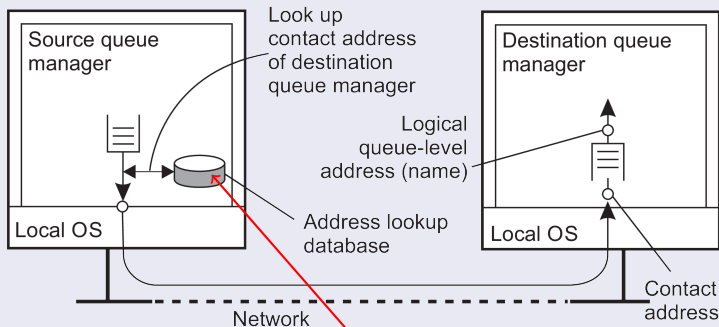
## MOM – Arquitetura geral



Via de regra, aplicações inserem (ou retiram) mensagens em filas locais (uma de entrada, e uma de saída)

# Comunicação Persistente

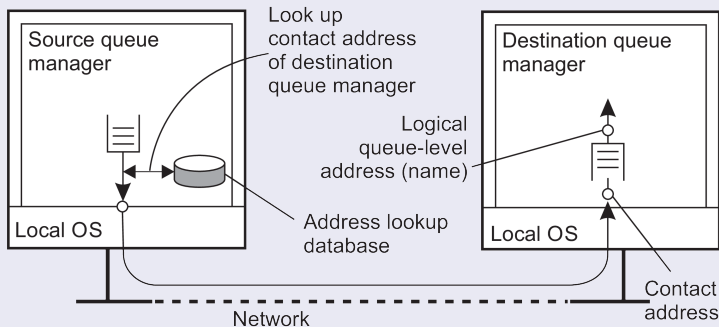
## MOM – Arquitetura geral



Cabe ao gerenciador fazer, com o auxílio de uma base de endereços, com que a mensagem chegue à fila de destino

# Comunicação Persistente

## MOM – Arquitetura geral



Alguns gerenciadores de fila podem operar como roteadores, transmitindo as mensagens a outros gerenciadores

## MOM – *Message Brokers*

- Sistemas de filas de mensagens assumem um **protocolo comum de troca de mensagens**
- Todas as aplicações usam o mesmo formato de mensagem (estrutura e representação de dados)



## MOM – *Message Brokers*

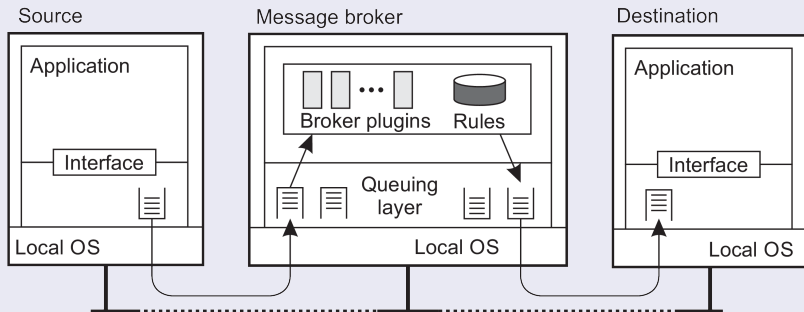
- Sistemas de filas de mensagens assumem um **protocolo comum de troca de mensagens**
  - Todas as aplicações usam o mesmo formato de mensagem (estrutura e representação de dados)
- Contudo, os sistemas (especialmente os legados) podem usar protocolos de comunicação distintos

## MOM – *Message Brokers*

- Sistemas de filas de mensagens assumem um **protocolo comum de troca de mensagens**
  - Todas as aplicações usam o mesmo formato de mensagem (estrutura e representação de dados)
- Contudo, os sistemas (especialmente os legados) podem usar protocolos de comunicação distintos
- Usamos então um broker de mensagem
  - Nós especiais encarregados de converter as mensagens para que possam ser entendidas pelo destinatário

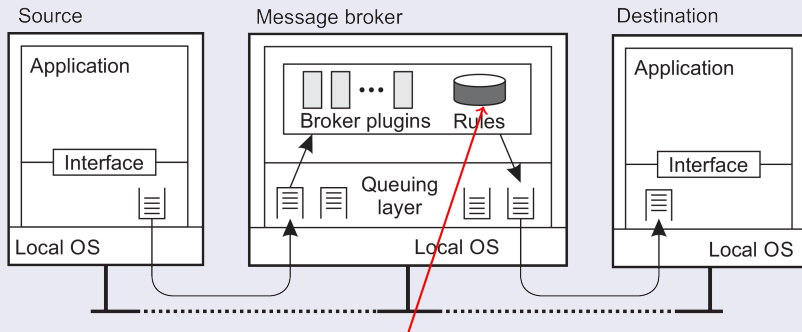
# Comunicação Persistente

## MOM – *Message Brokers*



# Comunicação Persistente

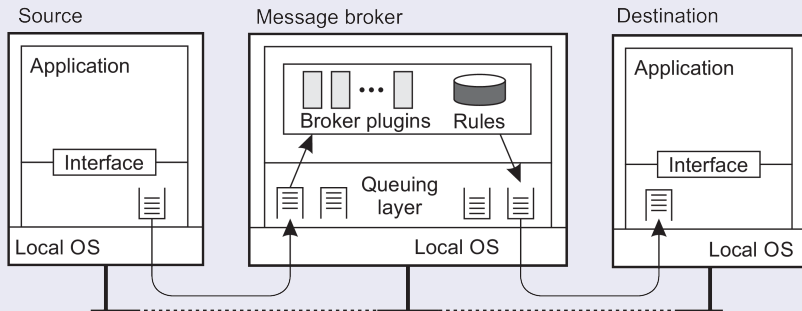
## MOM – *Message Brokers*



No coração do broker está um repositório de regras para transformar a mensagem de um tipo para outro

# Comunicação Persistente

## MOM – *Message Brokers*



Podem agir também como roteadores com base no conteúdo (em sistemas *publish-subscribe*)

# Comunicação Orientada a Mensagens

- Comunicação Transiente
- Comunicação Persistente
- **Comunicação Multicast**

# Comunicação *Multicast*

## *Broadcasting*

Quando a mensagem deve ser recebida por todos os nós da rede de *overlay*

# Comunicação *Multicast*

## *Broadcasting*

Quando a mensagem deve ser recebida por todos os nós da rede de *overlay*

## *Multicasting*

*Multicasting* se refere ao envio de uma mensagem a um subconjunto dos nós (um grupo específico destes)



# Comunicação *Multicast*

## *Broadcasting*

Quando a mensagem deve ser recebida por todos os nós da rede de *overlay*

## *Multicasting*

*Multicasting* se refere ao envio de uma mensagem a um subconjunto dos nós (um grupo específico destes)

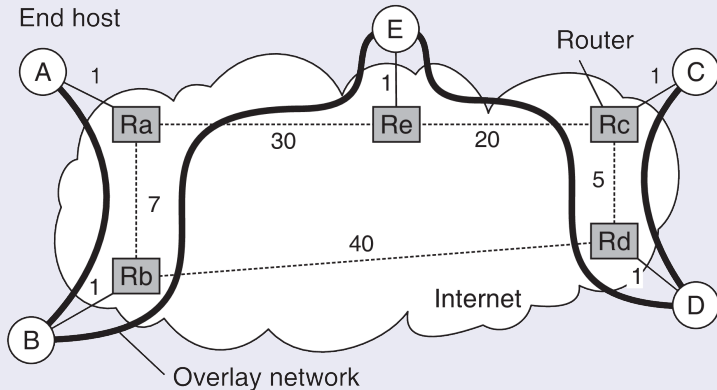
$$\textit{Multicasting} \subseteq \textit{Broadcasting}$$

## *Multicasting* no nível da aplicação

- Os nós do SD são organizados em uma **rede de overlay**, usada para disseminar dados
- Frequentemente uma árvore, levando assim a um único caminho entre cada par de nós
- Alternativamente, uma rede *mesh* (malha), em que cada nó tem múltiplos vizinhos
  - Pode haver múltiplos caminhos entre cada par de nós
  - Exige alguma forma de roteamento

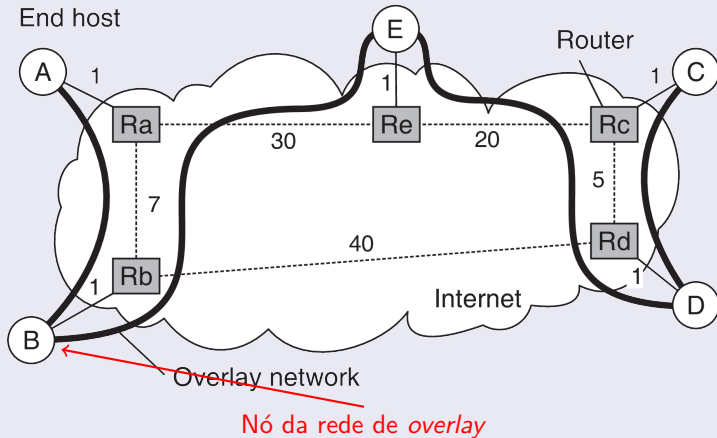
# Comunicação *Multicast*

## *Multicasting* no nível da aplicação: Custos



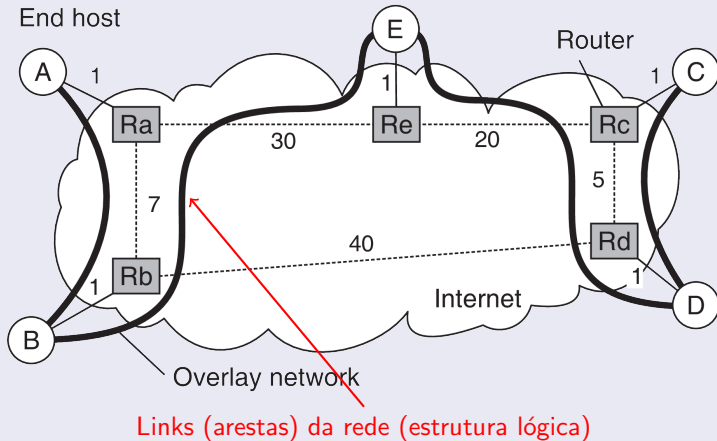
# Comunicação *Multicast*

## *Multicasting* no nível da aplicação: Custos



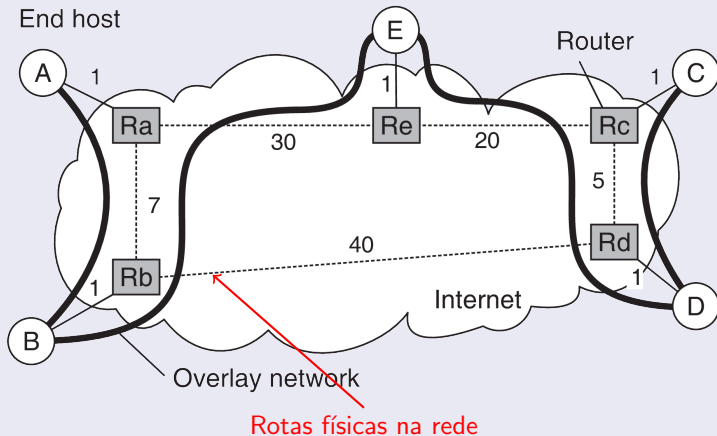
# Comunicação *Multicast*

## *Multicasting* no nível da aplicação: Custos



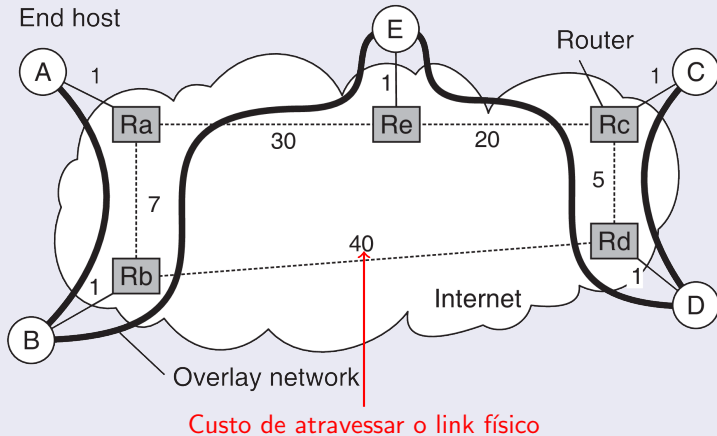
# Comunicação *Multicast*

## *Multicasting* no nível da aplicação: Custos



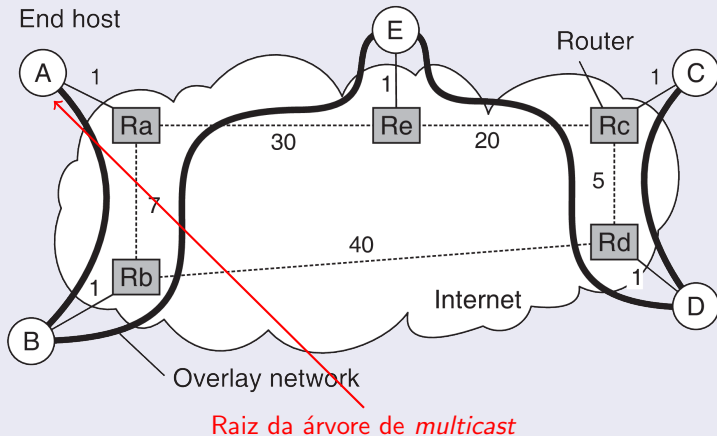
# Comunicação *Multicast*

## *Multicasting* no nível da aplicação: Custos



# Comunicação *Multicast*

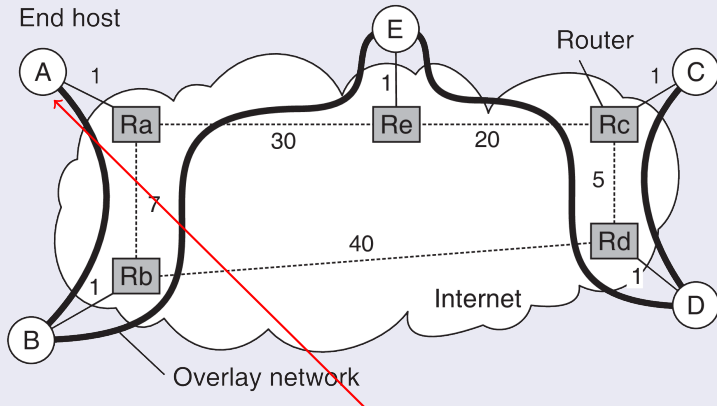
## *Multicasting* no nível da aplicação: Custos





# Comunicação *Multicast*

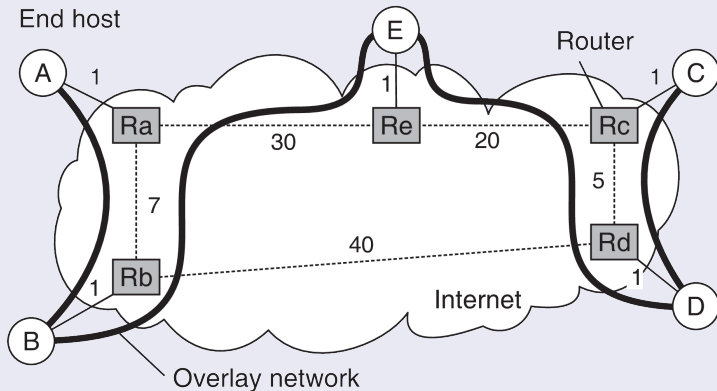
## Multicasting no nível da aplicação: Custos



Quando A faz o multicasting de uma mensagem, ele a envia a seus filhos, que a repassam a seus filhos, e assim por diante

# Comunicação *Multicast*

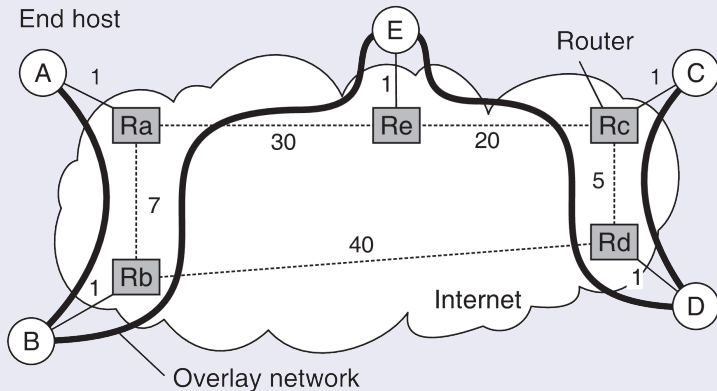
## *Multicasting* no nível da aplicação: Custos



O caminho físico será então  $A \rightarrow Ra, Ra \rightarrow Rb, Rb \rightarrow B, B \rightarrow Rb, Rb \rightarrow Ra, Ra \rightarrow Re, Re \rightarrow E, E \rightarrow Re, Re \rightarrow Rc, Rc \rightarrow Rd$  etc.

# Comunicação *Multicast*

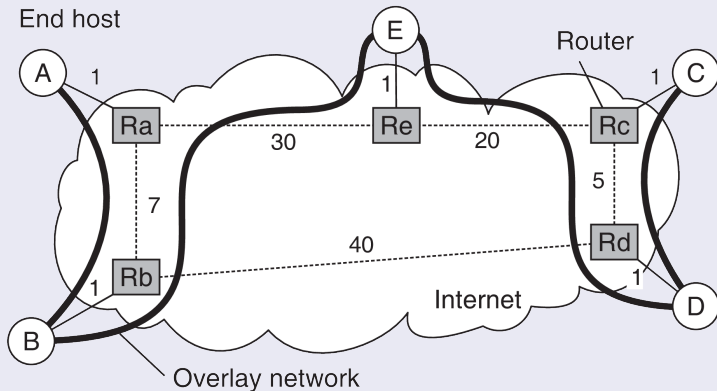
## *Multicasting* no nível da aplicação: Custos



E teremos passado 2 vezes por Rb → B,  
Rb → Ra, Re → E, Rd → D e Rd → Rc

# Comunicação *Multicast*

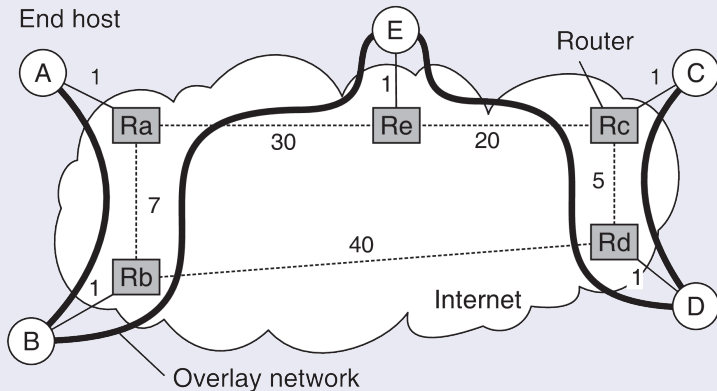
## *Multicasting* no nível da aplicação: Custos



Nossa rede de *overlay* (lógica) seria mais eficiente se os links (B,E) e (D,E) fossem substituídos por links (A,E) e (C,E)

# Comunicação *Multicast*

## *Multicasting* no nível da aplicação: Custos

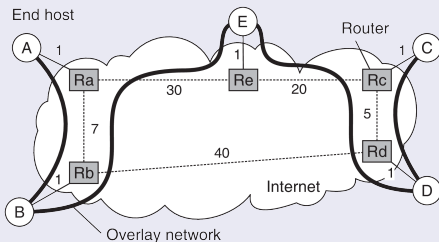


Embora agora passássemos 2 vezes por C → Rc, evitaríamos → B, Ra → Rb, D → Rd e Rc → Rd, reduzindo bastante o custo dos caminhos

## Árvore de MNA: Métricas de qualidade

- **Stress nos links**

- Com que frequência uma mensagem de *multicast* será enviada pelo mesmo enlace físico?
- Definida a cada link
- Um valor  $> 1$  indica que o mesmo link físico é compartilhado por diferentes links lógicos
- Ex: uma mensagem de A para D precisa atravessar  $\langle Ra, Rb \rangle$  duas vezes





## Árvore de MNA: Métricas de qualidade

- **Stretch**

- Ex: de B para C

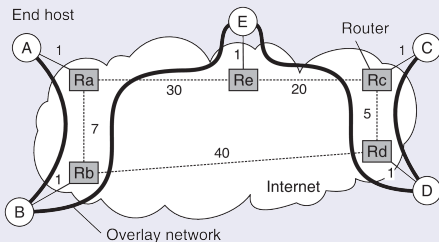
- *Overlay*:  $B \rightarrow Rb \rightarrow Ra \rightarrow Re \rightarrow E \rightarrow Re \rightarrow Rc \rightarrow Rd \rightarrow D \rightarrow Rd \rightarrow Rc \rightarrow C$

- Custo total: 73

- Rota física (sem considerar o *overlay*):  $B \rightarrow Rb \rightarrow Rd \rightarrow Rc \rightarrow C$

- Custo total: 47

- $stretch = 73/47 = 1,55$





## *Multicasting* baseado em inundação

- Cada nó simplesmente repassa a mensagem a cada um de seus vizinhos
  - Exceto para o vizinho de quem recebeu a mensagem
  - E somente se o nó não tiver visto a mensagem antes

## *Multicasting* baseado em inundação

- Cada nó simplesmente repassa a mensagem a cada um de seus vizinhos
  - Exceto para o vizinho de quem recebeu a mensagem
  - E somente se o nó não tiver visto a mensagem antes
- O método mais simples de fazer o *broadcasting* de uma mensagem
  - O método visto no último exemplo

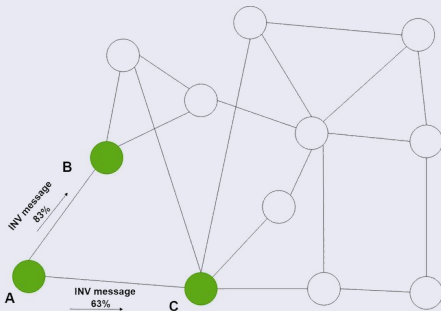
## *Multicasting* baseado em inundação

- Cada nó simplesmente repassa a mensagem a cada um de seus vizinhos
  - Exceto para o vizinho de quem recebeu a mensagem
  - E somente se o nó não tiver visto a mensagem antes
- O método mais simples de fazer o *broadcasting* de uma mensagem
  - O método visto no último exemplo
- Quanto mais arestas, mais caro o método, em termos de desempenho

# Comunicação *Multicast*

## *Multicasting* baseado em inundação

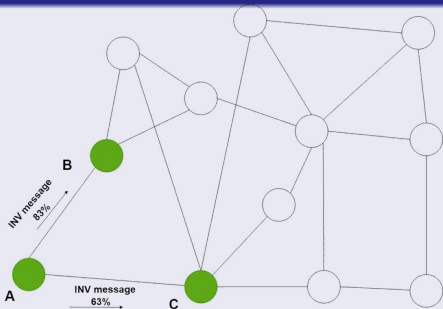
- Variação: **inundação probabilística**
  - Um nó repassa a mensagem a um vizinho com uma certa probabilidade  $p_{in}$
  - Podemos fazer com que a probabilidade dependa do número de vizinhos (o grau do nó), ou do grau de seus vizinhos



# Comunicação *Multicast*

## *Multicasting* baseado em inundação

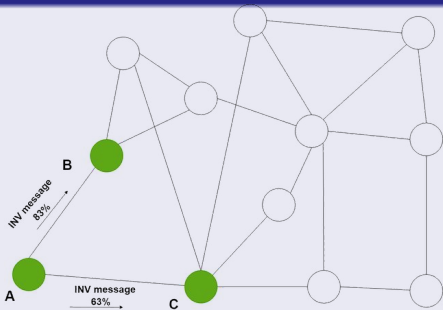
- Vantagem
  - O número total de mensagens enviadas cai linearmente com  $p_{in}$



# Comunicação *Multicast*

## *Multicasting* baseado em inundação

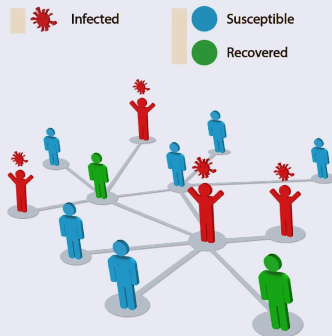
- Vantagem
  - O número total de mensagens enviadas cai linearmente com  $p_{in}$
- Desvantagem
  - Quanto menor  $p_{in}$ , maior a chance de nem todos os nós da rede serem atingidos



# Comunicação *Multicast*

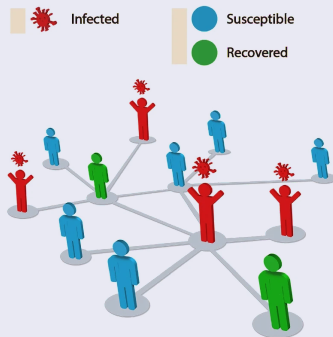
## Protocolos Epidêmicos

- Oriundos da observação de como doenças se espalham
- Também conhecido como fofoca (*gossip*)



## Protocolos Epidêmicos

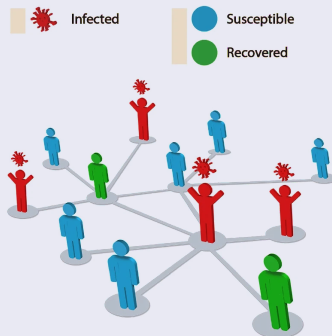
- Oriundos da observação de como doenças se espalham
- Também conhecido como fofoca (*gossip*)
- Buscam rapidamente propagar informação dentre um grande número de nós usando apenas informação local
- Não há componente central para coordenar a disseminação





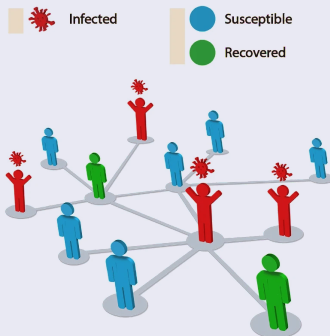
## Protocolos Epidêmicos: Nomenclatura

- Contaminado
  - Nó que contém dados a serem distribuídos



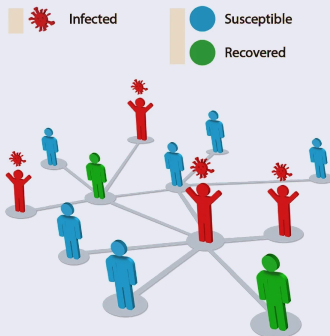
## Protocolos Epidêmicos: Nomenclatura

- Contaminado
  - Nó que contém dados a serem distribuídos
- Suscetível
  - Nó que ainda não viu esses dados



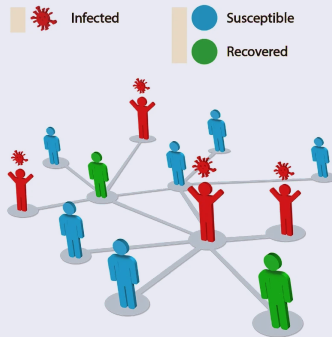
## Protocolos Epidêmicos: Nomenclatura

- Contaminado
  - Nó que contém dados a serem distribuídos
- Suscetível
  - Nó que ainda não viu esses dados
- Removido
  - Nó já atualizado, que não deseja ou é incapaz de espalhar seus dados



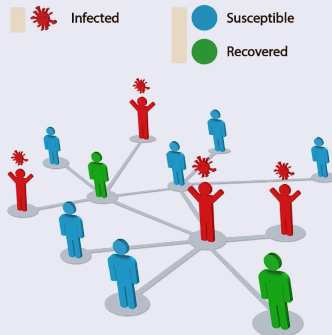
## Protocolos Epidêmicos: Modelos de propagação

- Anti-entropia
  - Cada nó P regularmente escolhe outro nó Q ao acaso e troca atualizações (uma determinada mensagem) com Q



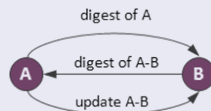
## Protocolos Epidêmicos: Modelos de propagação

- Anti-entropia
  - Cada nó P regularmente escolhe outro nó Q ao acaso e troca atualizações (uma determinada mensagem) com Q
- Espalhamento de boato (*rumour spreading*)
  - Um nó que acaba de ser atualizado (*contaminado*) repassa a atualização a alguns vizinhos (contaminando-os)



## Modelos de propagação: Anti-entropia

- Abordagens de atualização
  - **Push:** P só envia suas atualizações para Q
    - Má escolha – atualizações são propagadas apenas por nós infectados
    - Se muitos estiverem infectados, a probabilidade de um selecionar um nó suscetível é pequena
    - Um nó particular pode permanecer suscetível por um longo tempo



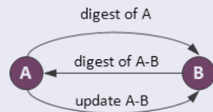
# Comunicação *Multicast*

## Modelos de propagação: Anti-entropia

- Abordagens de atualização

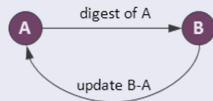
- Push:** P só envia suas atualizações para Q

- Má escolha – atualizações são propagadas apenas por nós infectados
- Se muitos estiverem infectados, a probabilidade de um selecionar um nó suscetível é pequena
- Um nó particular pode permanecer suscetível por um longo tempo



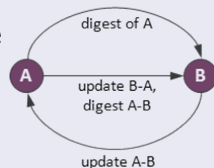
- Pull:** P só recebe atualizações de Q

- A propagação das atualizações é disparada pelos nós suscetíveis
- Funciona melhor quando muitos nós estão infectados – a chance é maior do nó contatar um contaminado



## Modelos de propagação: Anti-entropia

- **Push-Pull:** P e Q **trocaram** atualizações entre si (e acabam com as mesmas informações)
- Cada push-pull leva  $\mathcal{O}(\log(N))$  rodadas de comunicação para disseminar as atualizações para todos os  $N$  nós
- **Rodada:** quando cada nó já tomou a iniciativa de iniciar uma troca com outro nó escolhido aleatoriamente

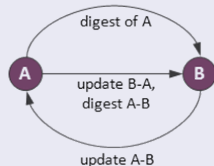




# Comunicação *Multicast*

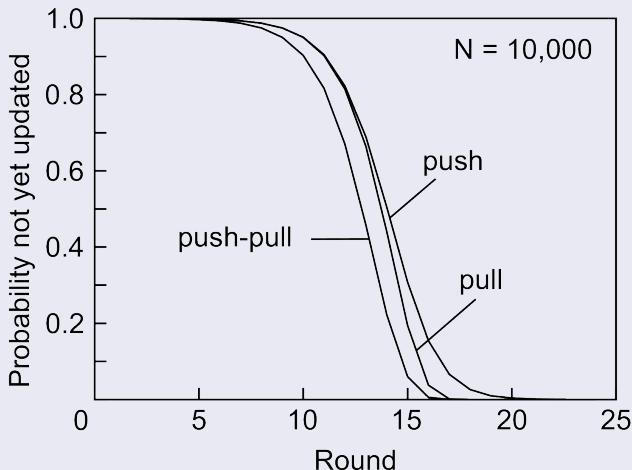
## Modelos de propagação: Anti-entropia

- **Push-Pull:** P e Q **trocam** atualizações entre si (e acabam com as mesmas informações)
- Cada push-pull leva  $\mathcal{O}(\log(N))$  rodadas de comunicação para disseminar as atualizações para todos os  $N$  nós
- **Rodada:** quando cada nó já tomou a iniciativa de iniciar uma troca com outro nó escolhido aleatoriamente
- Se somente um nó estiver contaminado, as atualizações rapidamente se espalharão por todos os nós com qualquer uma das formas de anti-entropia
- Embora push-pull continue sendo a melhor estratégia



# Comunicação *Multicast*

## Modelos de propagação: Anti-entropia



## Modelos de propagação: *Rumour spreading*

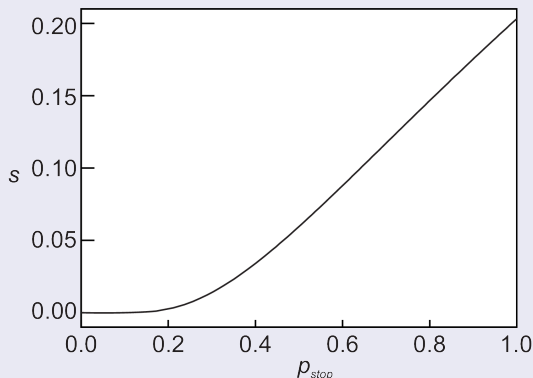
- Funcionamento
  - Se o nó P foi atualizado com algum item x, ele contacta um nó arbitrário Q e tenta atualizar Q com x
  - Se, contudo, Q já tiver sido atualizado, P pode perder o interesse em espalhar x (ou seja, contatar outros nós), com uma determinada probabilidade  $p_{parar}$
  - Se transforma em removido, em nossa nomenclatura

## Modelos de propagação: *Rumour spreading*

- Funcionamento
  - Se o nó P foi atualizado com algum item x, ele contacta um nó arbitrário Q e tenta atualizar Q com x
  - Se, contudo, Q já tiver sido atualizado, P pode perder o interesse em espalhar x (ou seja, contatar outros nós), com uma determinada probabilidade  $p_{parar}$ 
    - Se transforma em removido, em nossa nomenclatura
- Não há garantias de que todo nó será atualizado
  - A fração de nós que não receberão a atualização (suscetíveis) é dada por  $s = e^{-(1/p_{parar}+1)(1-s)}$

# Comunicação *Multicast*

## Modelos de propagação: *Rumour spreading*

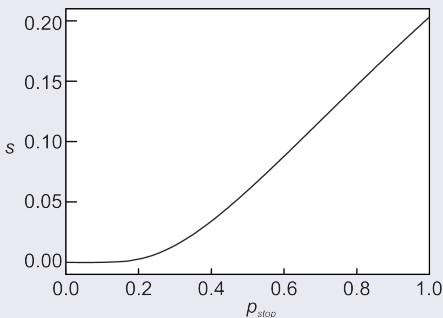


Considere 10.000 nós		
$1/p_{parar}$	$s$	$N_s$
1	0.203188	2032
2	0.059520	595
3	0.019827	198
4	0.006977	70
5	0.002516	25
6	0.000918	9
7	0.000336	3

Mesmo para valores altos de  $P_{parar}$ , a fração  $s$  de nós ignorantes é relativamente baixa

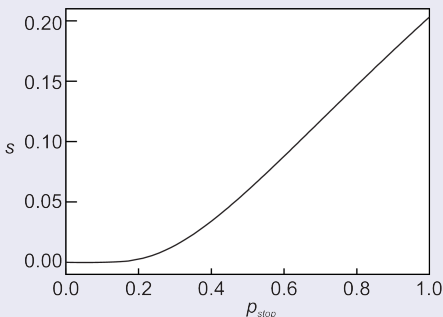
## Modelos de propagação: *Rumour spreading*

- Vantagem:
  - Escalabilidade, dado o baixo número de sincronizações entre nós



## Modelos de propagação: *Rumour spreading*

- Vantagem:
  - Escalabilidade, dado o baixo número de sincronizações entre nós
- Desvantagens:
  - Cada nó deve conhecer todos os outros
  - Se quisermos nos assegurar de que todos os nós serão atualizados, *rumour spreading* sozinho não é suficiente



## Protocolos Epidêmicos: Removendo dados

- Complicado...
  - Se simplesmente removermos o dado de um nó, ele acabará recebendo as cópias antigas, interpretando como atualizações de algo que não possuía antes
  - E o dado volta ao nó, desfazendo a remoção



## Protocolos Epidêmicos: Removendo dados

- Complicado...
  - Se simplesmente removermos o dado de um nó, ele acabará recebendo as cópias antigas, interpretando como atualizações de algo que não possuía antes
  - E o dado volta ao nó, desfazendo a remoção
- Como “propagar a remoção” então?

## Protocolos Epidêmicos: Removendo dados

- Complicado...
  - Se simplesmente removermos o dado de um nó, ele acabará recebendo as cópias antigas, interpretando como atualizações de algo que não possuía antes
  - E o dado volta ao nó, desfazendo a remoção
- Como “propagar a remoção” então?
  - O truque é registrar a remoção como uma atualização qualquer

## Protocolos Epidêmicos: Removendo dados

- Complicado...
  - Se simplesmente removermos o dado de um nó, ele acabará recebendo as cópias antigas, interpretando como atualizações de algo que não possuía antes
  - E o dado volta ao nó, desfazendo a remoção
- Como “propagar a remoção” então?
  - O truque é registrar a remoção como uma atualização qualquer
  - Registramos então a remoção propagando um **atestado de óbito** do dado em questão

## Protocolos Epidêmicos: Removendo dados

- E como remover o atestado de óbito?
  - Ele não pode ficar lá pra sempre

## Protocolos Epidêmicos: Removendo dados

- E como remover o atestado de óbito?
  - Ele não pode ficar lá pra sempre
- Possibilidades
  - Execute um algoritmo global para detectar se a remoção foi percebida por todos os nós e então remova os atestados
  - Assuma que os atestados não serão propagados para sempre e associe um tempo de vida máximo para o atestado (correndo o risco dele não ter sido propagado a todos os nós)

## Protocolos Epidêmicos: Removendo dados

- É preciso que a remoção alcance todos os nós
  - Fazemos alguns nós manterem atestados de óbito que nunca são descartados

## Protocolos Epidêmicos: Removendo dados

- É preciso que a remoção alcance todos os nós
  - Fazemos alguns nós manterem atestados de óbito que nunca são descartados
- Suponha que o nó  $P$  tem um atestado para  $x$ 
  - Se alguma atualização obsoleta de  $x$  chegar a  $P$ , este pode reagir, espalhando o atestado novamente