第4讲 进程通信

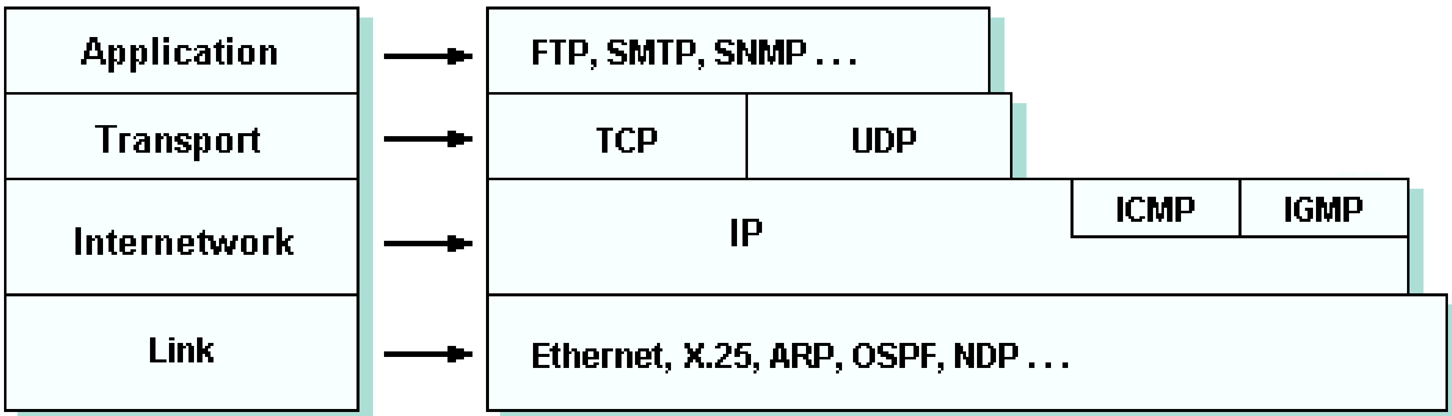
§4.1 远程过程调用

§4.2 面向消息的通信

§4.3 流通信与多播通信

基于网络通信

- TCP/IP协议



- 通信类型
  - 瞬态通信：消息需被即时传递和接收，不存储
  - 持久通信：消息存储在通信服务器直到被接收
  - 同步通信、异步通信

通信中间件

- 提供附加的、通用性的服务功能
  - 包含丰富的（应用层）通信协议；
  - 包装/解包装数据，对于系统集成非常重要；
  - 命名协议，允许资源的共享；
  - 扩展机制，例如复制和缓存。

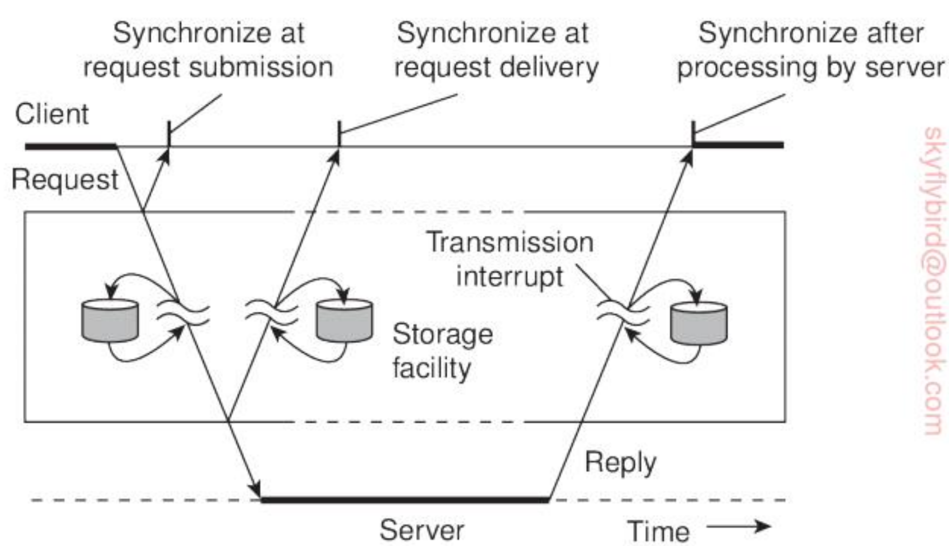
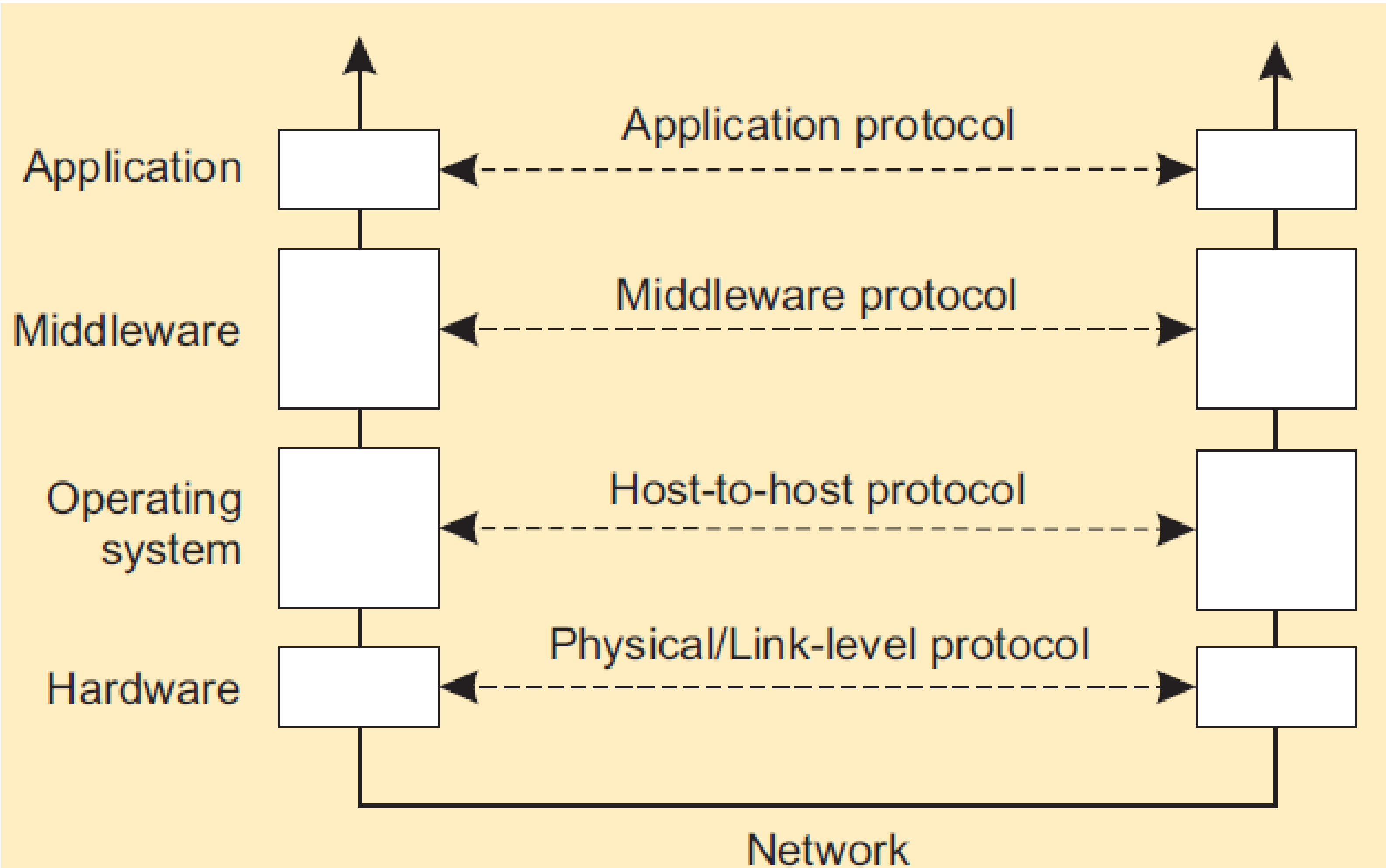


Figure 4.4: Viewing middleware as an intermediate (distributed) service in application-level communication.

分布式系统通信模型



Application protocol
Application
Middleware protocol
Middleware
Host-to-host protocol
Operating system
Physical/Link-level protocol
Hardware
Network

客户端/服务器通信

- 客户/服务器一般是基于瞬态、同步的通信方法：
  - 客户和服务器在通信时必须处于活跃的状态；
  - 客户端发出请求后，被阻塞直到收到应答；
  - 服务器只是等待到来的请求，然后处理这些请求。
- 同步通信的缺点：
  - 客户端等待回应的时候不能做其他工作；
  - 失效必须即刻处理。

§4.1 远程过程调用

- Remote Procedure Call (RPC)
- RPC allows a program to
  - transparently call procedures located on another machine.
  - No message passing is visible to the programmer.
- A widely used technique that underlies many distributed systems.

Conventional Procedure Call

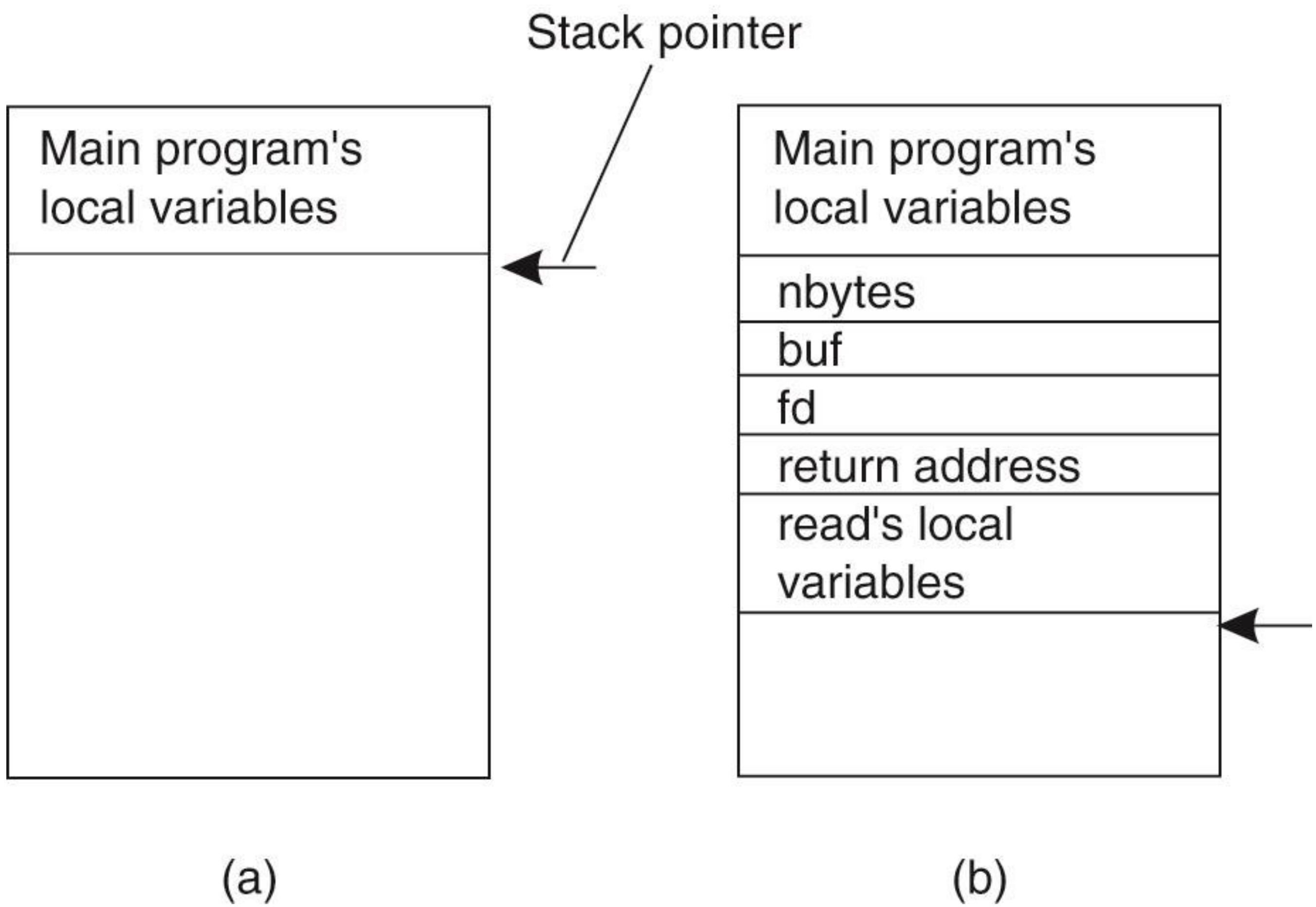


Figure 4-5. (a) Parameter passing in a local procedure call: the stack before the call to read.
(b) The stack while the called procedure is active.

Client and Server Stubs

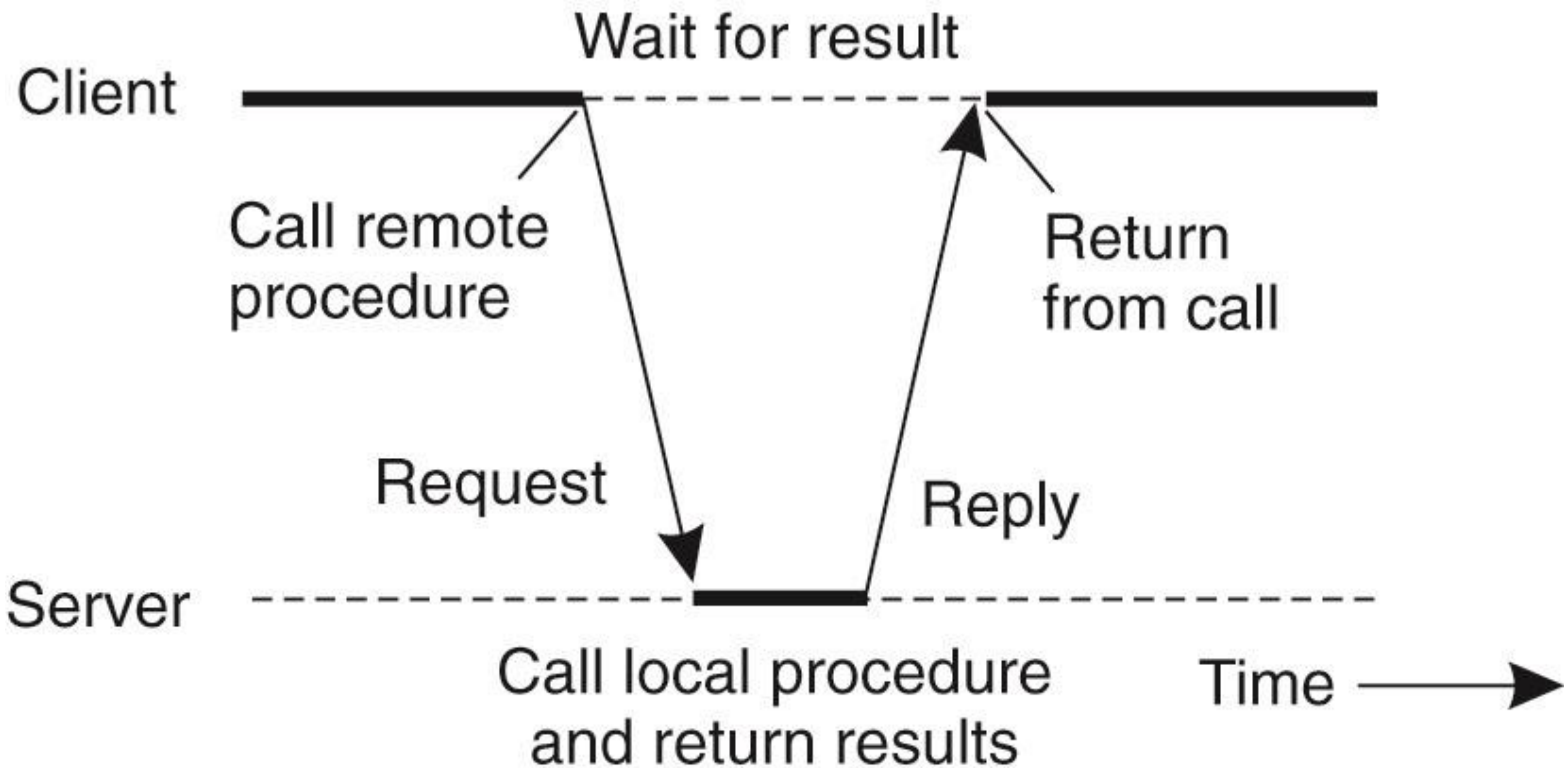
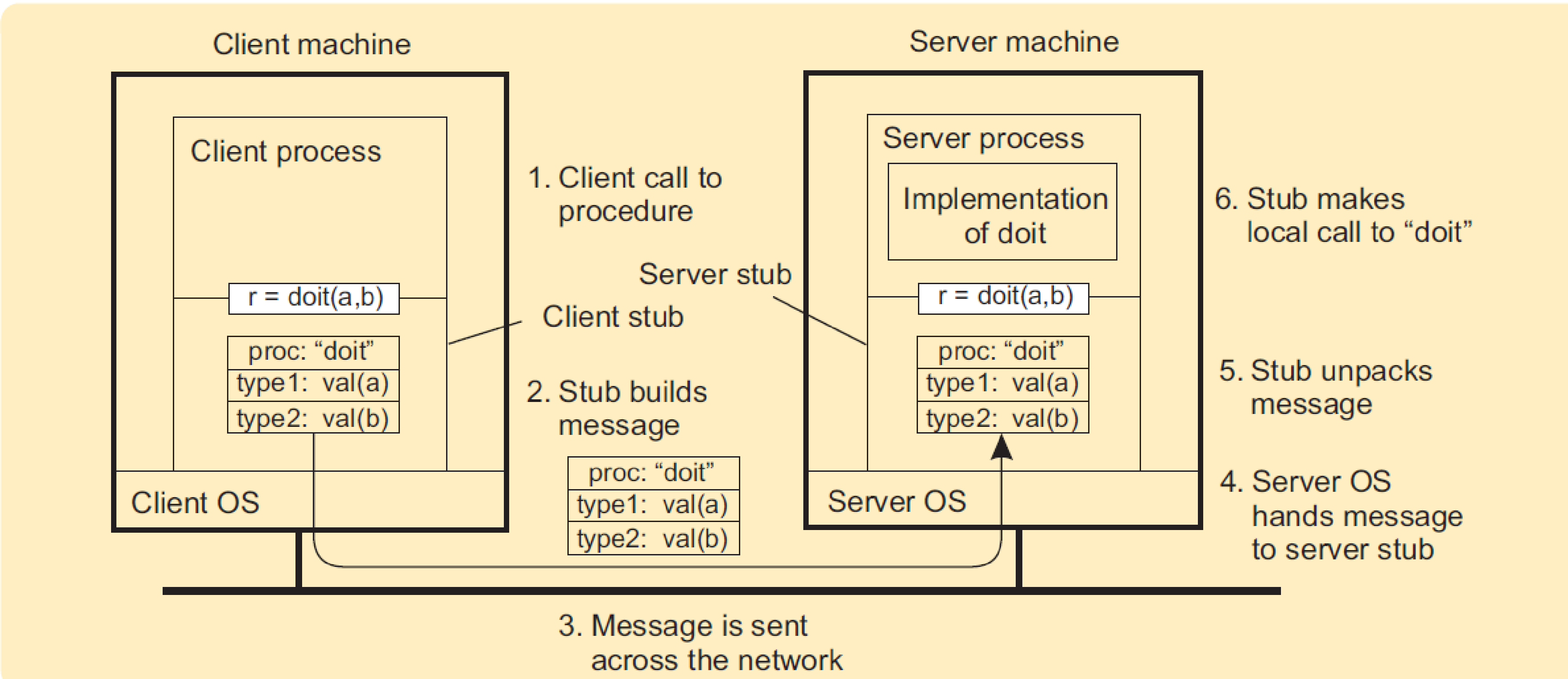


Figure 4-6. Principle of RPC between a client and server program.

Remote Procedure Calls



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters; calls server.
6. Server does local call; returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result; returns to client.

Parameter Passing in RPC

- At the client:
  - packing parameters into a message (*parameter marshaling*)
- At the server:
  - re-interpreted the parameters from message
- Tricky/Complex
  - Machines may use different character codes.
    - EBCDIC versus ASCII versus Unicode
  - Different representation of integers and floating-point numbers
  - Different byte addressing: little-endian versus big-endian format

Pass-by-reference?

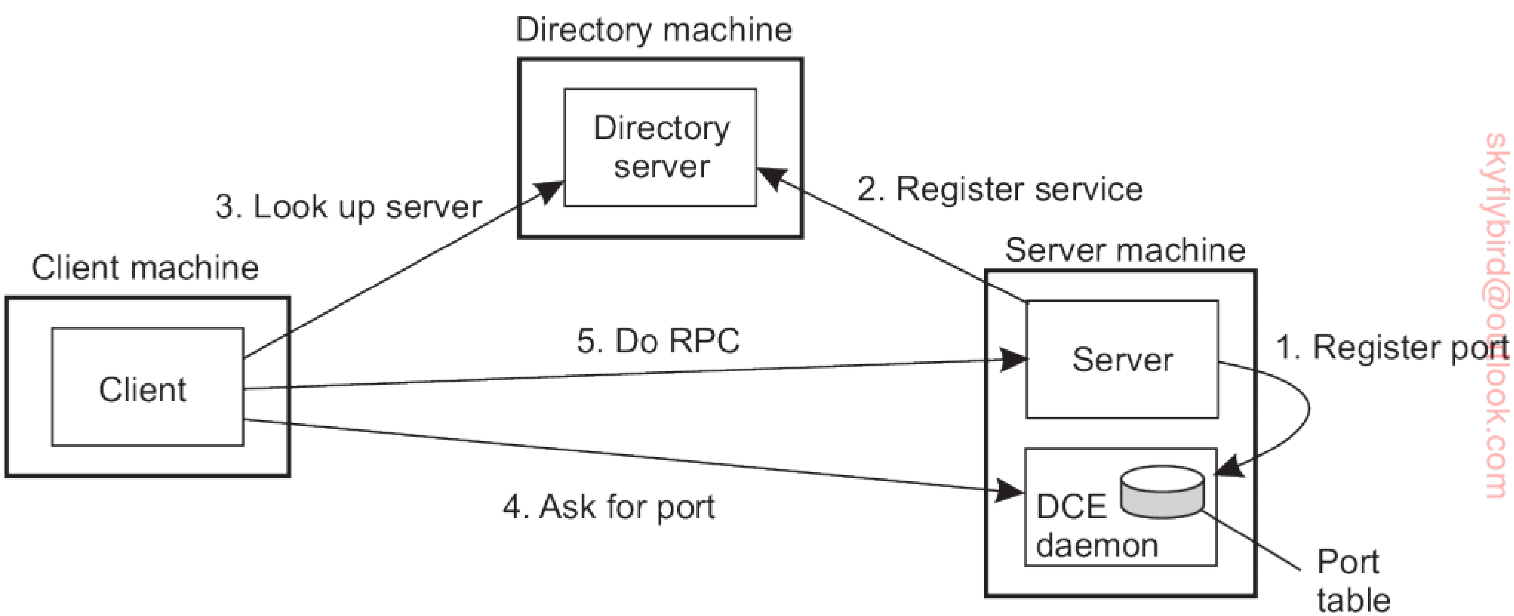
Parameter Passing in RPC

```
import channel, pickle

class Client:
  def append(self, data, dbList):
    msglst = (APPEND, data, dbList)          # message payload
    msgsnd = pickle.dumps(msglst)            # wrap call
    self.chan.sendTo(self.server, msgsnd)    # send request to server
    msgrcv = self.chan.recvFrom(self.server) # wait for response
    retval = pickle.loads(msgrcv[1])         # unwrap return value
    return retval                            # pass it to caller

class Server:
  def run(self):
    while True:
      msgreq = self.chan.recvFromAny() # wait for any request
      client = msgreq[0]               # see who is the caller
      msgrpc = pickle.loads(msgreq[1]) # unwrap the call
      if APPEND == msgrpc[0]:          # check what is being requested
        result = self.append(msgrpc[1], msgrpc[2]) # do local call
        msgres = pickle.dumps(result)              # wrap the result
        self.chan.sendTo([client],msgres)          # send response
```

Figure 4.9: A simple RPC example for operation append, but now with proper marshaling.

Client-Server Binding

- One approach： directory server
  - First： Locate the server’ s machine.
  - Second： Locate the server (i.e., the process) on that machine.



skyflybird@outlook.com

Figure 4.17: Client-to-server binding in DCE.

Programming Client/Server

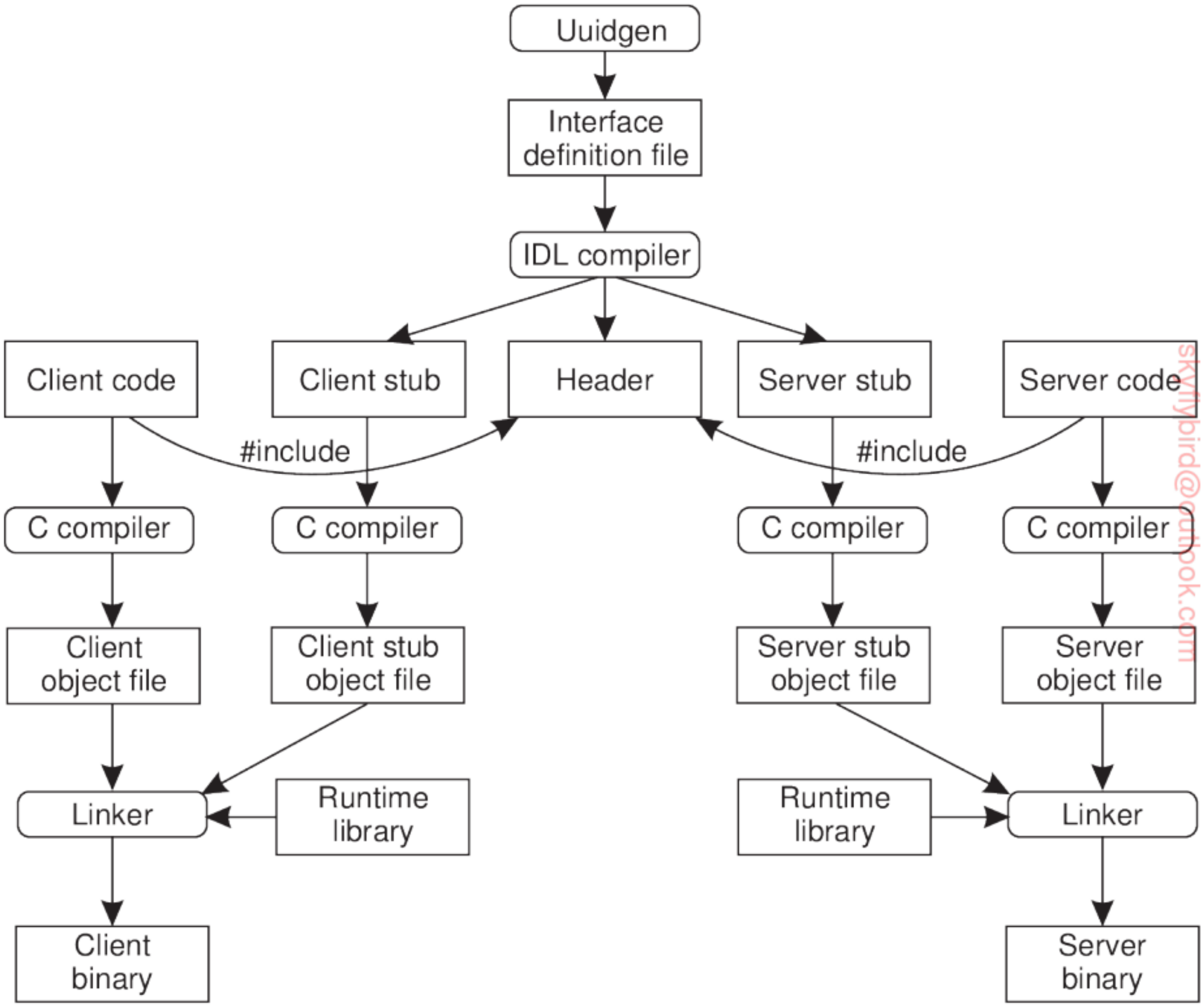


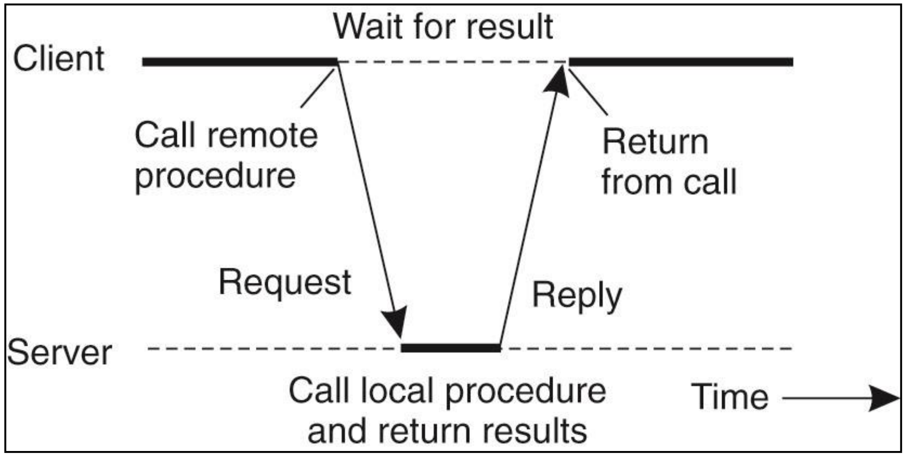
Figure 4.16: The steps in writing a client and a server in DCE RPC.

skyflybird@outlook.com

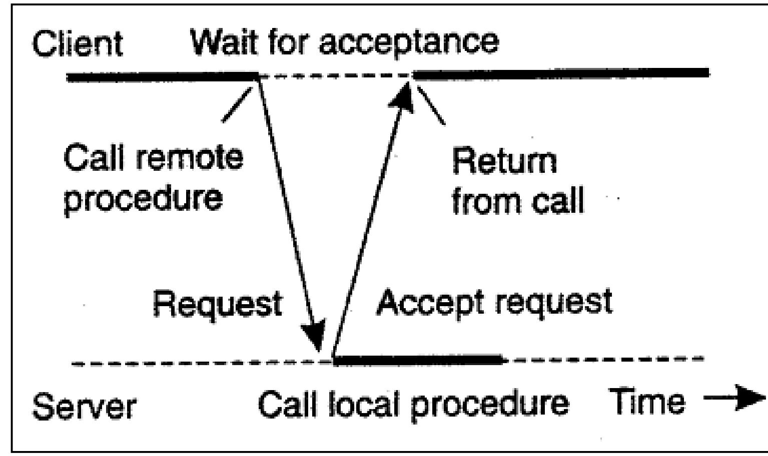
Asynchronous RPC

- Asynchronous RPC: (for calls no explicit results)
  - A client immediately continues once the server accepts the request (after an ack message from the server)
  - The server executes the RPC request after sending ack
- Deferred asynchronous RPC: (for calls with results back)
  - The client calls the server with a RPC request and the server immediately acknowledges it.
  - Later the server does a callback to the client with the result.
- One-way RPC:
  - Asynchronous RPC without ack

Asynchronous RPC

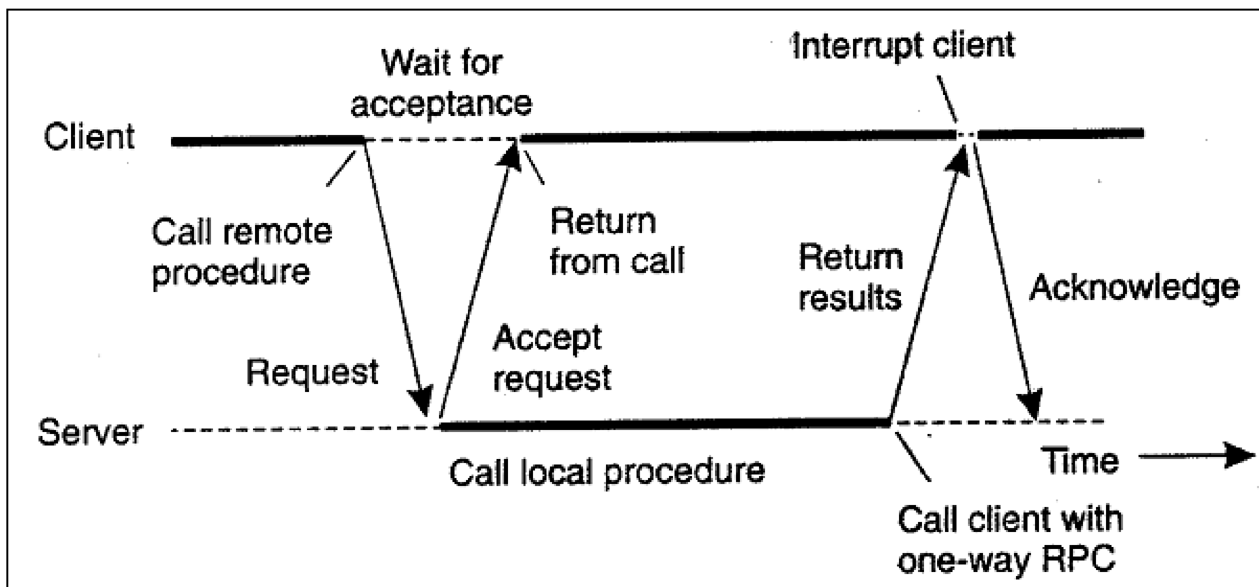


Common RPC



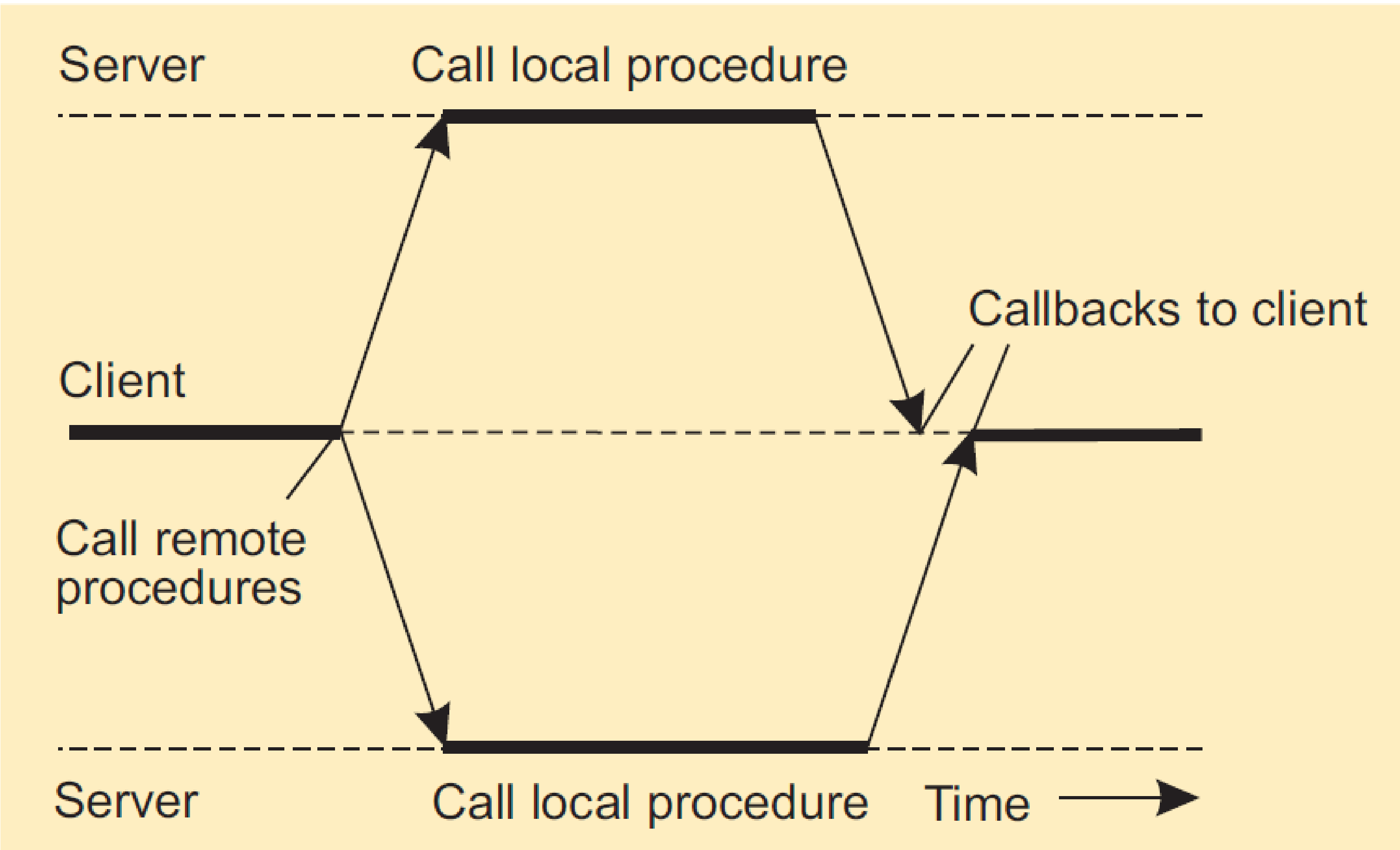
Asynchronous RPC

Deferred asynchronous RPC



多播RPC

- 发送RPC请求到一组服务器上



RPC Implementations

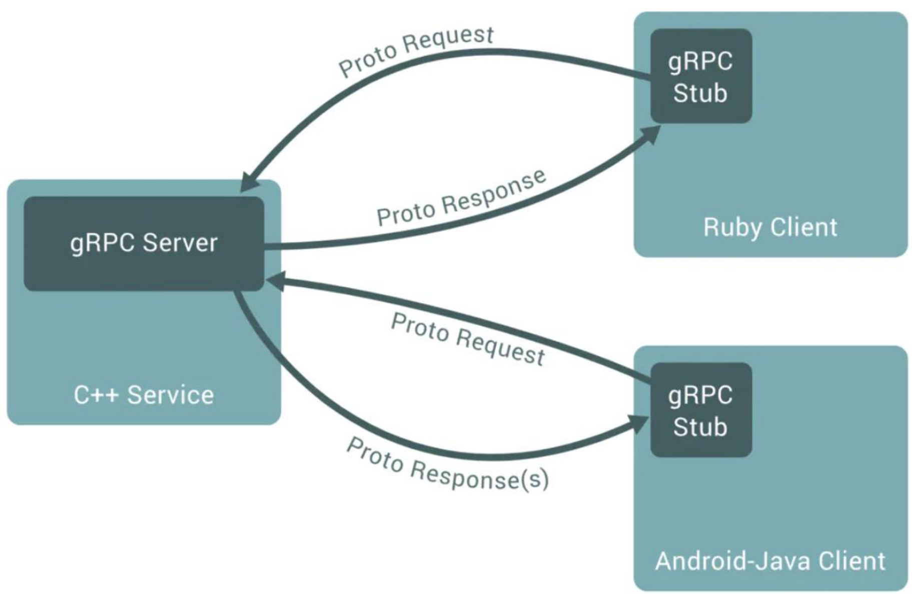
- Distributed Computing Environment / Remote Procedure Calls (DCE/RPC)
  - by the Open Software Foundation (now Open Group)
  - adopted in Microsoft’s base for distributed computing, DCOM.
- Open Network Computing Remote Procedure Call (ONC RPC)
  - originally developed by Sun Microsystems as part of their Network File System
  - Also called Sun ONC or Sun RPC
  - widely deployed remote procedure call system

RPC Style Implementations

- Java RMI (Remote Method Invocation)
- XML-RPC and its successor SOAP:
  - An RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism
- Microsoft .NET Remoting:
  - offers RPC/RMI facilities for distributed systems implemented on the Windows platform
- Facebook's Thrift protocol and framework

Google gRPC

- 基于Protbuf数据交换协议
  - 语言无关、平台无关
    - 支持 Java、C++、Python 等多种语言，支持多个平台
  - 高效: 比 XML 小(3~10倍)、快(20~100倍)、简单
  - 扩展性、兼容性好:
    - 可以更新数据结构，而不影响和破坏原有的旧程序



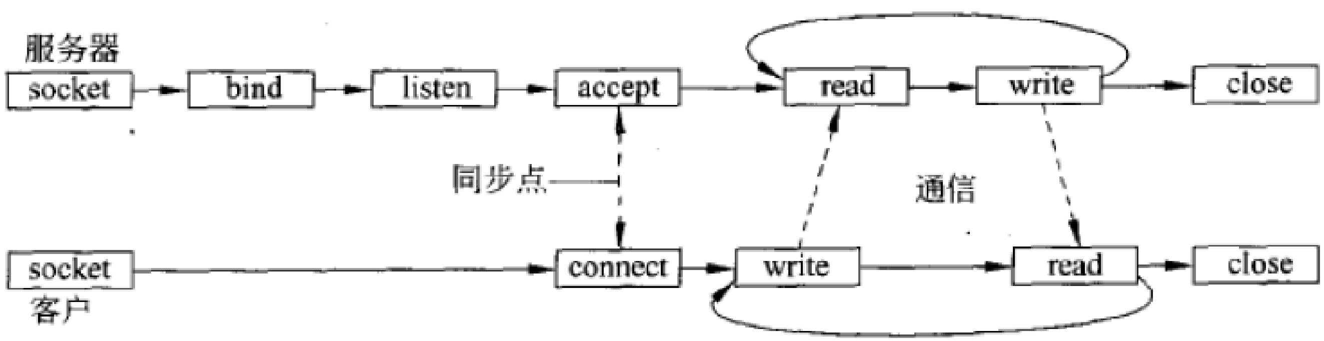
§4.2 面向消息的通信

- Transient
  - Socket-based message passing
  - ZeroMQ: Implementing messaging patterns
  - Message-Passing Interface (MPI)
- Persistent
  - Message queuing

Berkeley Socket

- The socket primitives for TCP/IP.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



ZeroMQ：简化Sockets使用

- 套接字不够方便
  - 属于底层编程，编程麻烦且容易出错
- 一些套接字的使用模式非常类似
  - 可以进行抽象实现
- ZeroMQ
  - 利用“配对”sockets提供高层次的表达
  - 实现常用的消息交换模式
    - 支持Many-to-one：一个server监听多个port
  - 所有的通信都是异步的：TCP+异步
    - Sender发送消息后不阻塞等待
    - 可以在接收方setup之前就发送
  - 关键点：后台线程负责排队和管理连接

ZeroMQ

- 实现三种常用模式
  - Request- Reply
  - Pub-sub
  - Pipeline

```
import zmq
context = zmq.Context()

p1 = "tcp://"+ HOST +":"+ PORT1 # how and where to connect
s  = context.socket(zmq.REQ)    # create request socket

s.connect(p1)                   # block until connected
s.send("Hello world 1")         # send message
message = s.recv()              # block until response
s.send("STOP")                  # tell server to stop
print message                   # print result
```

Figure 4.22: (b) A ZeroMQ client-server system: the client.

The request-reply pattern:
-绑定了接收动作与后续动作:
-即：server的send会自动定向为所收到的消息的sender

```
import zmq
context = zmq.Context()

p1 = "tcp://"+ HOST +":"+ PORT1 # how and where to connect
p2 = "tcp://"+ HOST +":"+ PORT2 # how and where to connect
s  = context.socket(zmq.REP)    # create reply socket

s.bind(p1)                      # bind socket to address
s.bind(p2)                      # bind socket to address
while True:
  message = s.recv()            # wait for incoming message
  if not "STOP" in message:     # if not to stop...
    s.send(message + "*")       # append "*" to message
  else:                         # else...
    break                       # break out of loop and end
```

Figure 4.22: (a) A ZeroMQ client-server system based: the server.

ZeroMQ

- Publish-subscribe
- Socket type
  - PUB, SUB

```
import zmq, time

context = zmq.Context()
s = context.socket(zmq.PUB)        # create a publisher socket
p = "tcp://"+ HOST +":"+ PORT      # how and where to communicate
s.bind(p)                          # bind socket to the address
while True:
  time.sleep(5)                    # wait every 5 seconds
  s.send("TIME " + time.asctime()) # publish the current time
```

Figure 4.23: (a) A multicasting socket-based time server

实现一对多的多播:
-消息会发送给所有匹配的client

```
import zmq

context = zmq.Context()
s = context.socket(zmq.SUB)          # create a subscriber socket
p = "tcp://"+ HOST +":"+ PORT        # how and where to communicate
s.connect(p)                         # connect to the server
s.setsockopt(zmq.SUBSCRIBE, "TIME")  # subscribe to TIME messages

for i in range(5):  # Five iterations
  time = s.recv()   # receive a message
  print time
```

Figure 4.23: (b) A client for the multicasting socket-based time server.

ZeroMQ

- Pipeline pattern
- Socket type:
  - PUSH, PULL

```
import zmq, time, pickle, sys, random

context = zmq.Context()
me   = str(sys.argv[1])
s    = context.socket(zmq.PUSH)          # create a push socket
src = SRC1  if me == ’1’ else SRC2     # check task source host
prt = PORT1 if me == ’1’ else PORT2    # check task source port
p    = "tcp://"+ src +":"+ prt          # how and where to connect
s.bind(p)                              # bind socket to address

for i in range(100):                   # generate 100 workloads
  workload = random.randint(1, 100)    # compute workload
  s.send(pickle.dumps((me,workload))) # send workload to worker
```

Figure 4.24: (a) A task simulating the generation of work.

Push与pull可以是多对多：
-匹配策略：先到先得
-task并发放进pipe
-worker轮流得到task

```
import zmq, time, pickle, sys

context = zmq.Context()
me = str(sys.argv[1])
r  = context.socket(zmq.PULL)       # create a pull socket
p1 = "tcp://"+ SRC1 +":"+ PORT1     # address first task source
p2 = "tcp://"+ SRC2 +":"+ PORT2     # address second task source
r.connect(p1)                       # connect to task source 1
r.connect(p2)                       # connect to task source 2

while True:
  work = pickle.loads(r.recv())     # receive work from a source
  time.sleep(work[1]*0.01)          # pretend to work
```

Figure 4.24: (b) A worker task.

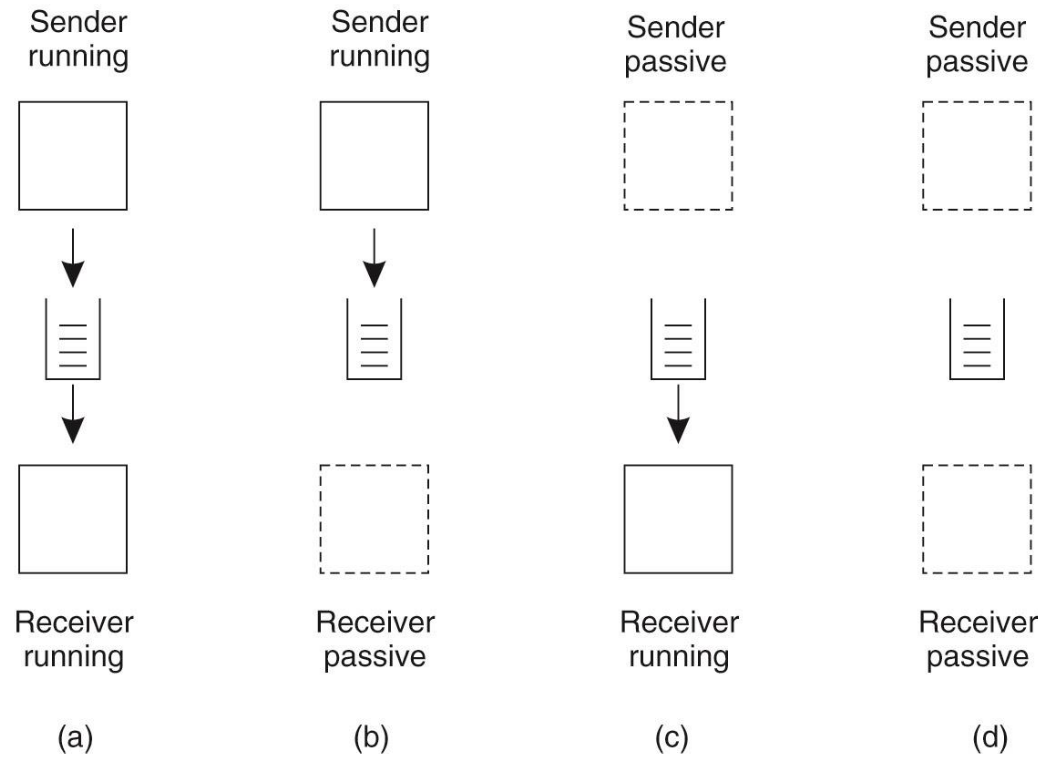
Message-Passing Interface (MPI)

- 面向高性能的专用网络
  - 支持不同协议栈：TCP/IP、专用协议
  - 高性能：考虑各种缓冲与同步机制
  - 更丰富的操作、灵活性好
- 已知的通信实体：(groupID, processID)

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Message Queuing System

- Persistent Communication



应用程序仅将消息放在本地队列中，然后由队列管理者将消息路由到其他地方。

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

一般架构：Queue level addressing

- Queue manager + local queue+ application
- 如何发送和交付消息？三个技术问题：
  - 目的队列的地址？逻辑地址与位置无关
    - Name-address (host+port)，放在一个表/库中
  - QM如何得到name-address映射信息？
    - 配置然后复制，简单但是低效、维护开销大（不一致性）

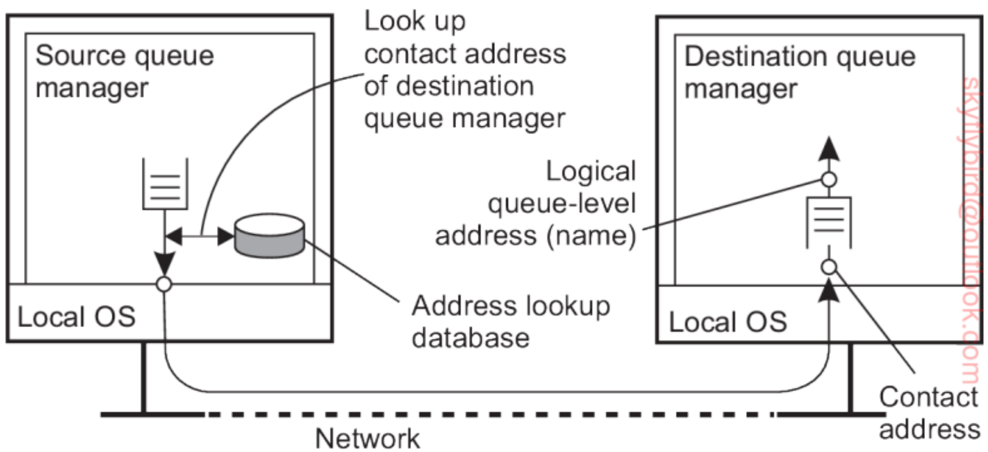
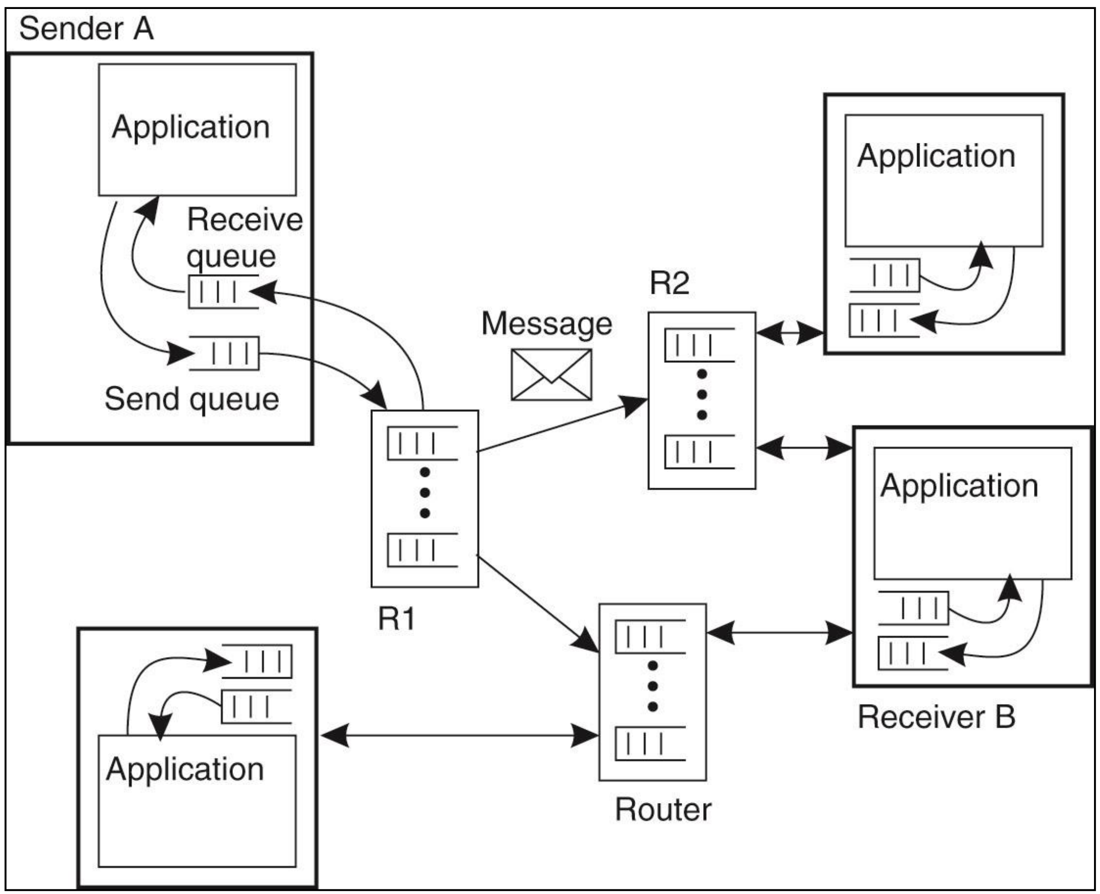


Figure 4.28: The relationship between queue-level naming and network-level addressing.

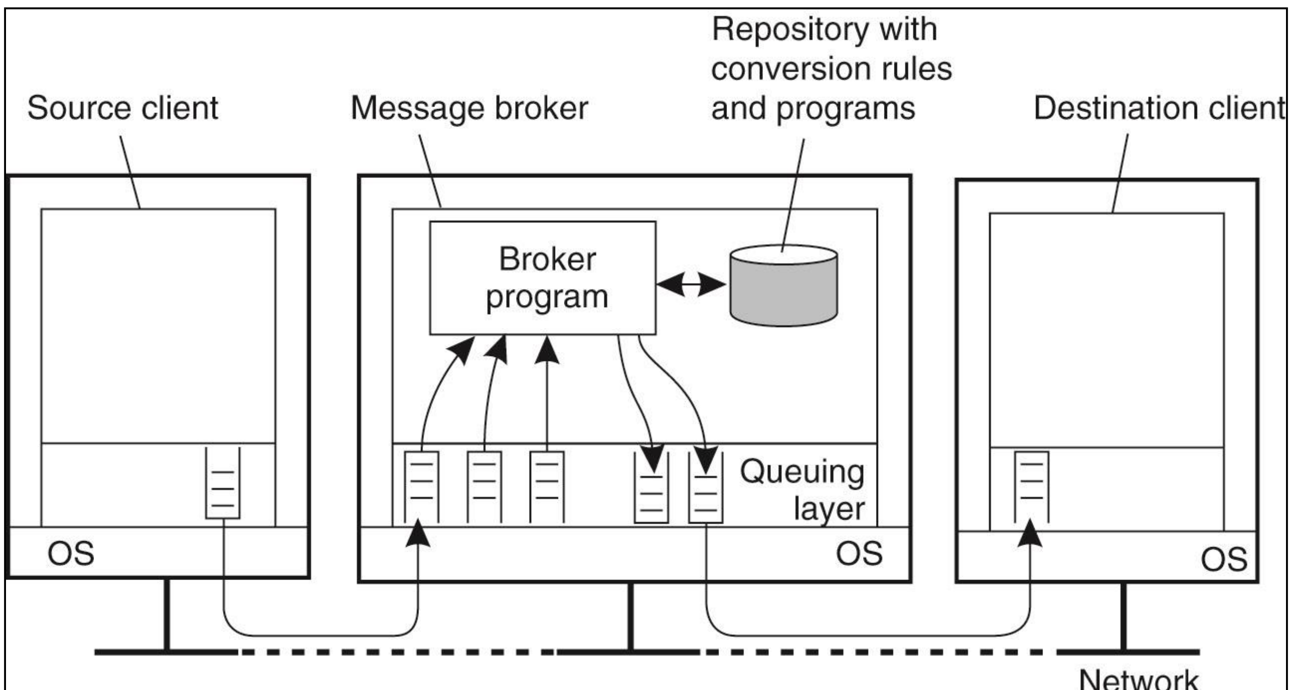
消息路由架构

- 消息路由器进行消息路由、转发
- 通过路由机制分发name-address匹配信息

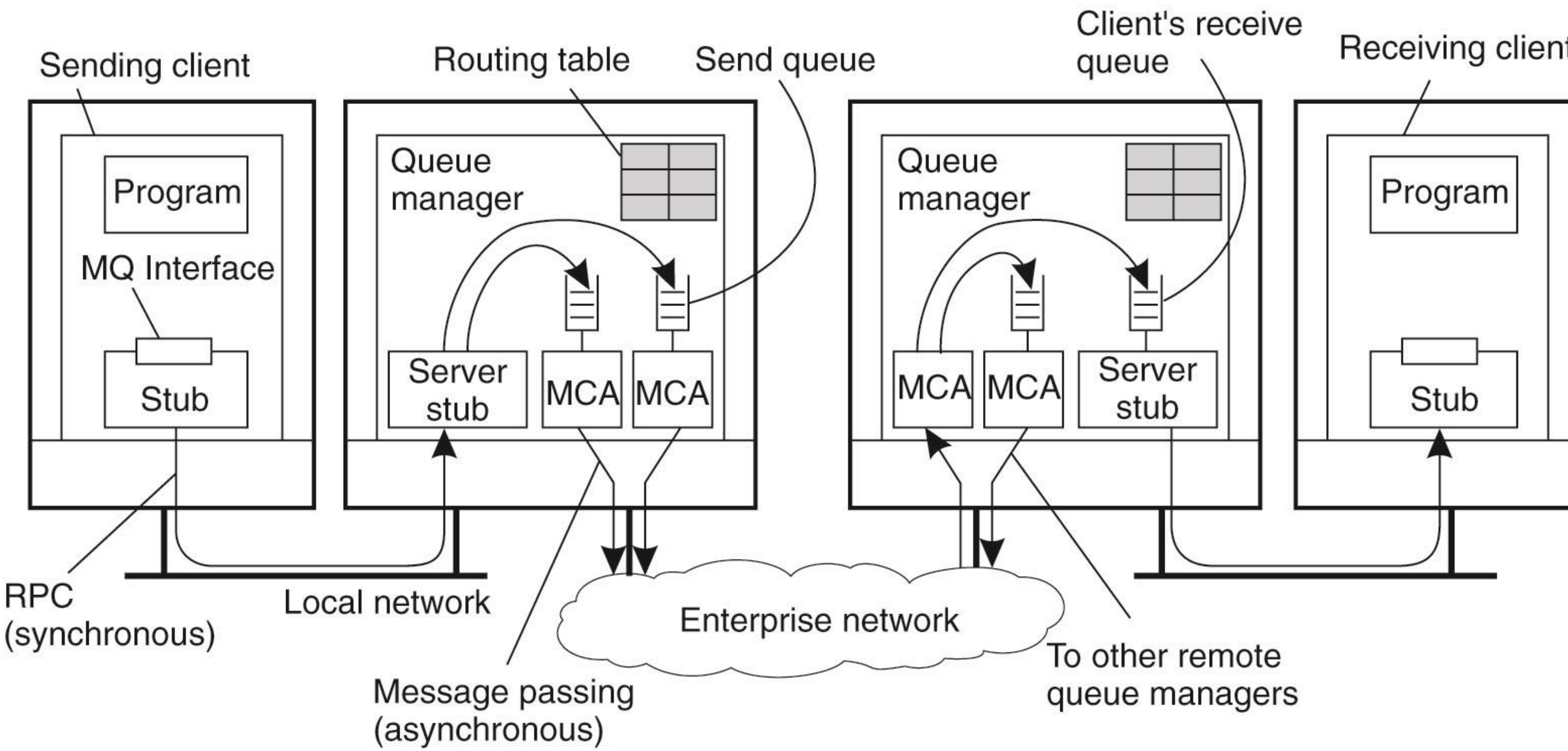


消息代理（转换器）

- 转换器处理应用消息的异构性
  - 将输入消息转换成目的格式
  - 起应用层网关的作用
  - 提供基于主题的路由功能（pub-sub）



IBM WebSphere MQ System



Channel: 不同进程的队列间的通信信道
MCA: message channel agent

MCA

- 主要作用：
  - 利用底层的网络通信协议如TCP/IP等建立通信信道
  - 从输出（输入）的网络传输包中封装（解封装）消息
  - 发送和接收传输包
- 属性：
  - 每个MCA都有一组相关的属性，这些属性决定了通道的全部特性

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

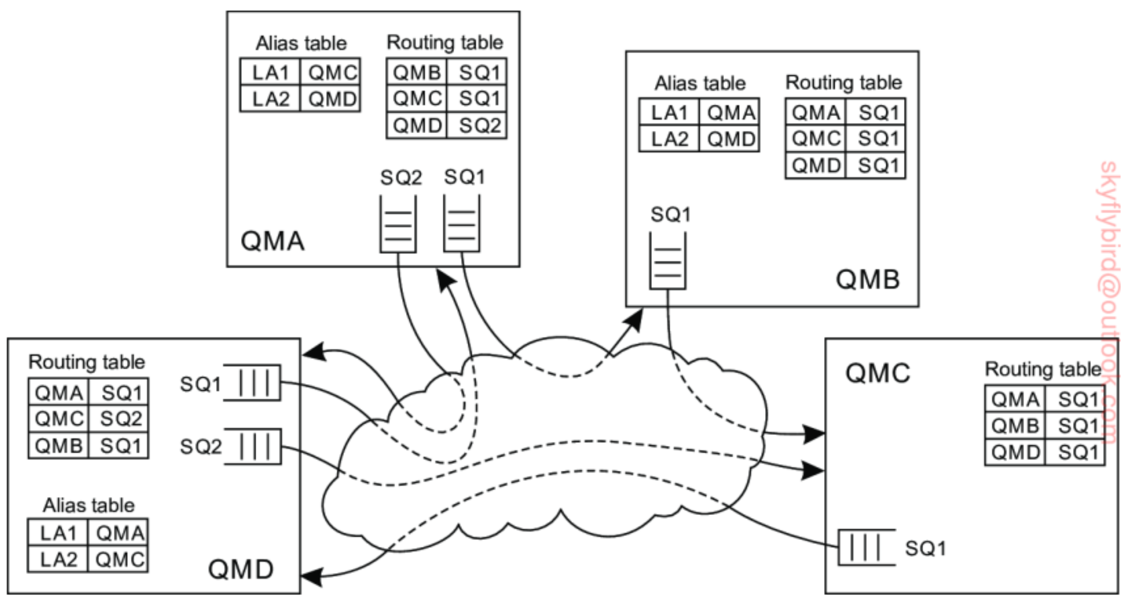
Figure 4.31: Some attributes associated with message channel agents.

编程接口

Primitive	Description
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

路由

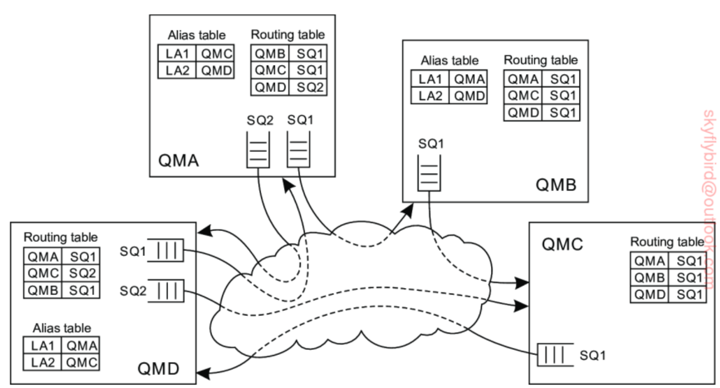
- 路由被显式地存储在队列管理器中的路由表中。
- 路由表的条目为（destQM, sendQ）对。
- 路由表中的条目称为别名。



skyflybird@outlook.com

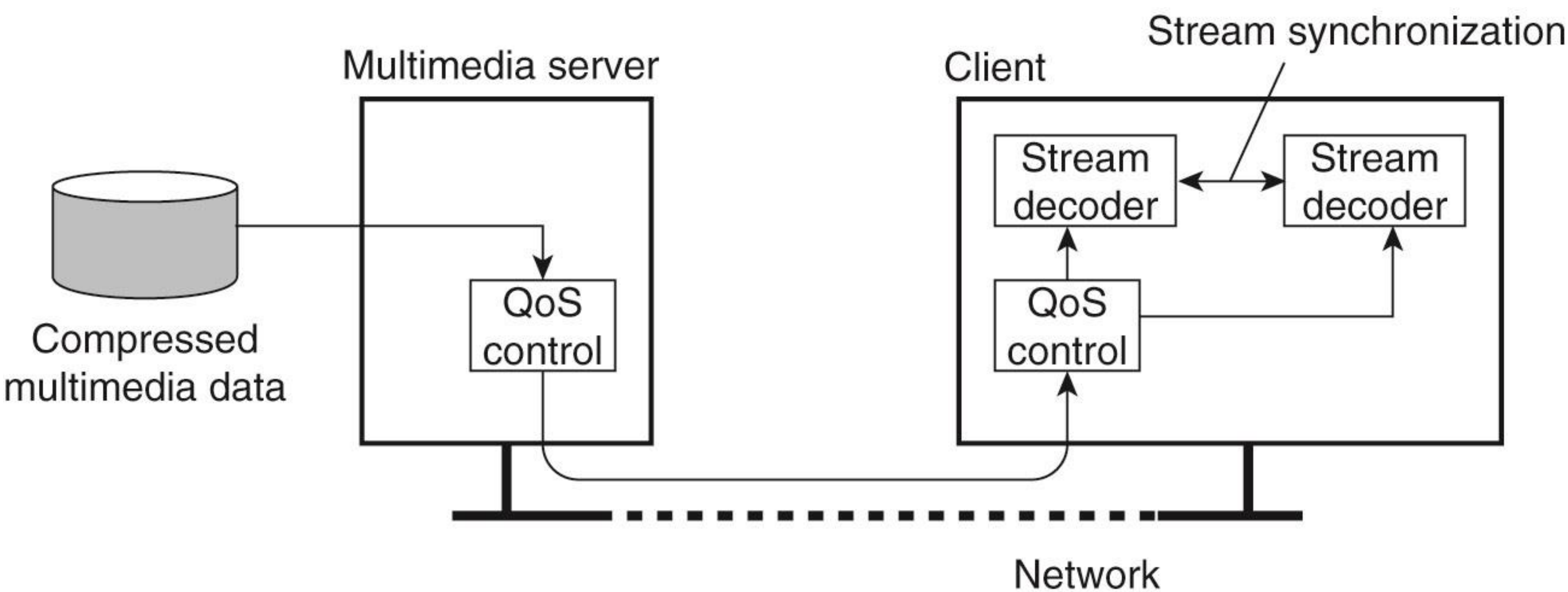
路由

- Managing overlay networks
  - To connect distributed queue managers into a consistent overlay network, and
  - maintain it over time
- How? Manually by administrators.
  - This administration not only involves creating channels between queue managers, but also
  - filling in the routing tables.



§4.3流通信与多播通信

- Data stream: time-dependent information
- 传输模式：异步、同步、等时



A general architecture for streaming stored multimedia data over a network.

Streams and Quality of Service (QoS)

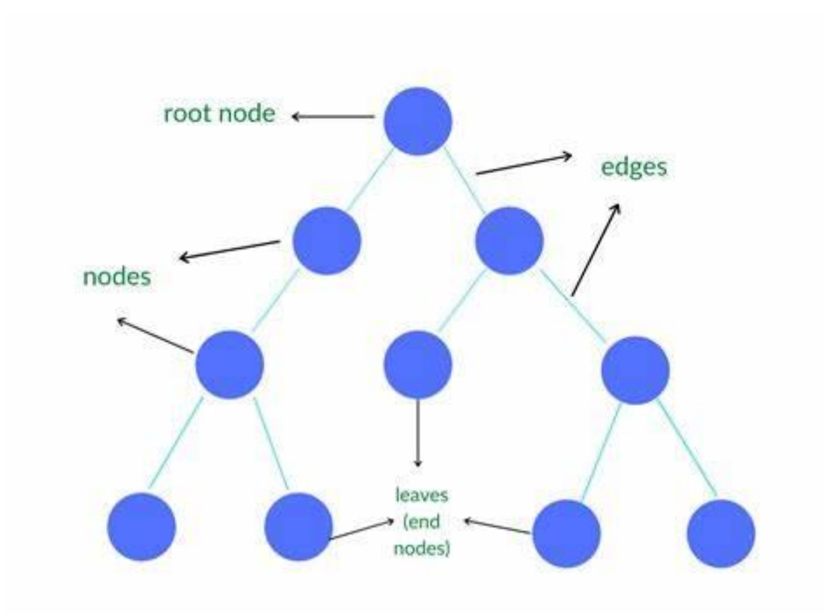
- Properties for Quality of Service:
  - The required bit rate at which data should be transported.
  - The maximum delay until a session has been set up
  - The maximum end-to-end delay .
  - The maximum delay variance, or jitter.
  - The maximum round-trip delay.
- Techniques enforcing QoS
  - Buffer
  - Coding
  - Frame interleaving

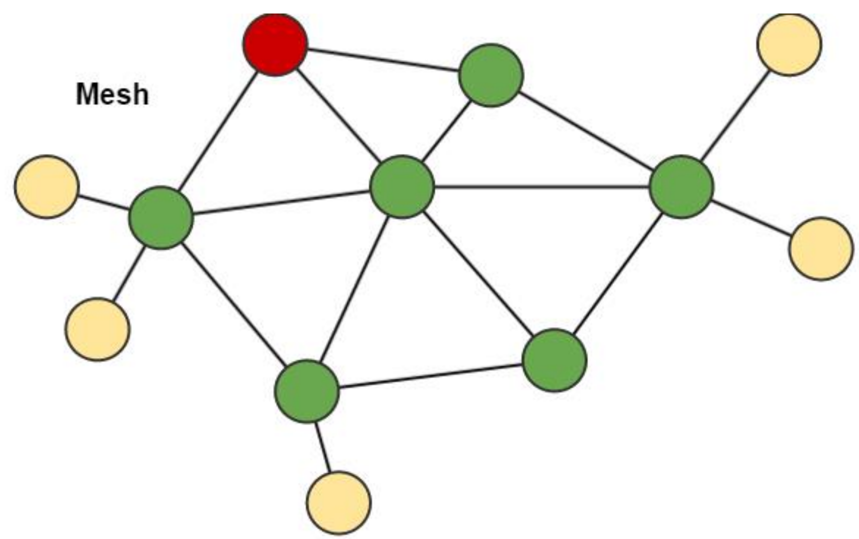
Multicast Comm.

- Network protocol support multicast
  - Too costly
  - Usually not supported
- Application-level multicast
  - Overlay-based
  - Gossip-based

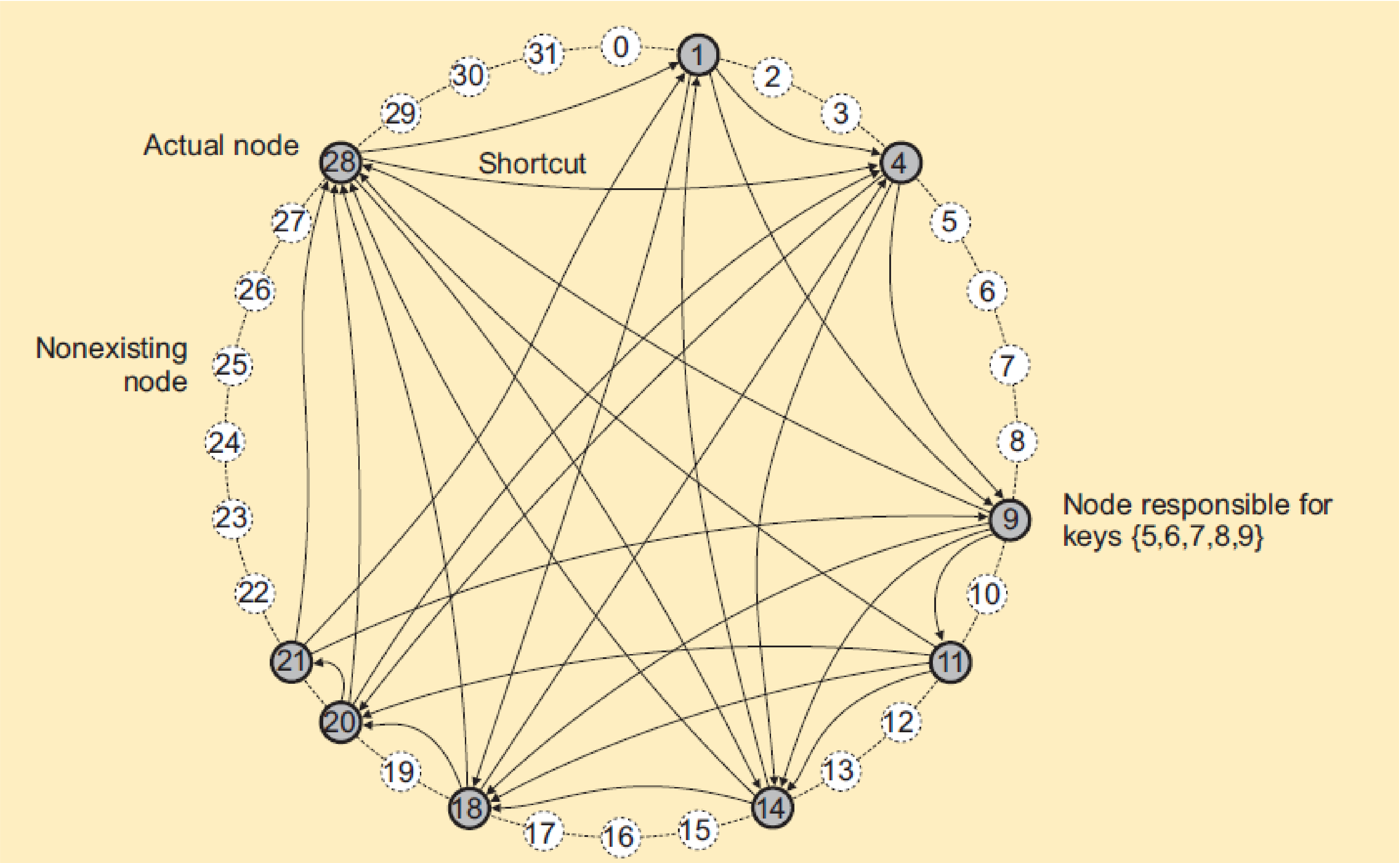
Overlay-based Multicast

- Overlay network, 覆盖网络
  - Communication path → virtual link
  - Multicast along the overlay
- Structure: tree or mesh
  - cost vs robustness





基于Chord的树

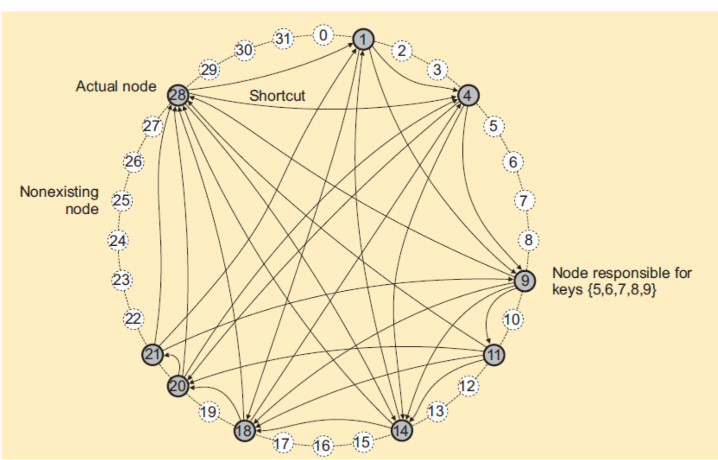


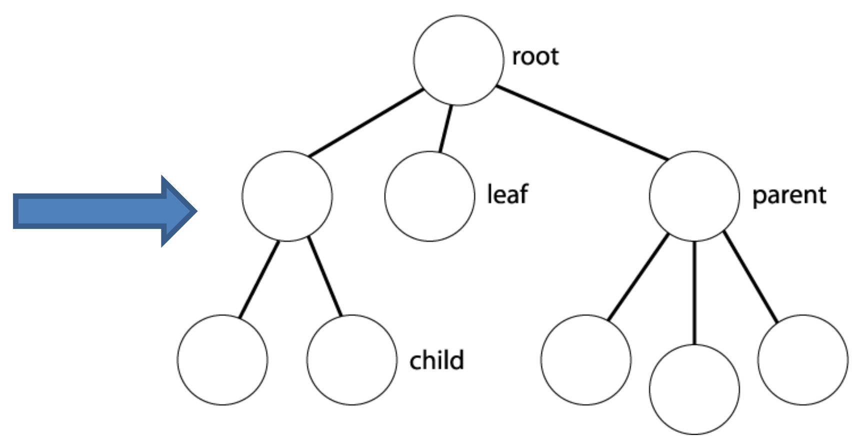
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
Actual node
Shortcut
Nonexisting node
Node responsible for keys {5,6,7,8,9}

基于Chord的树

Basic approach

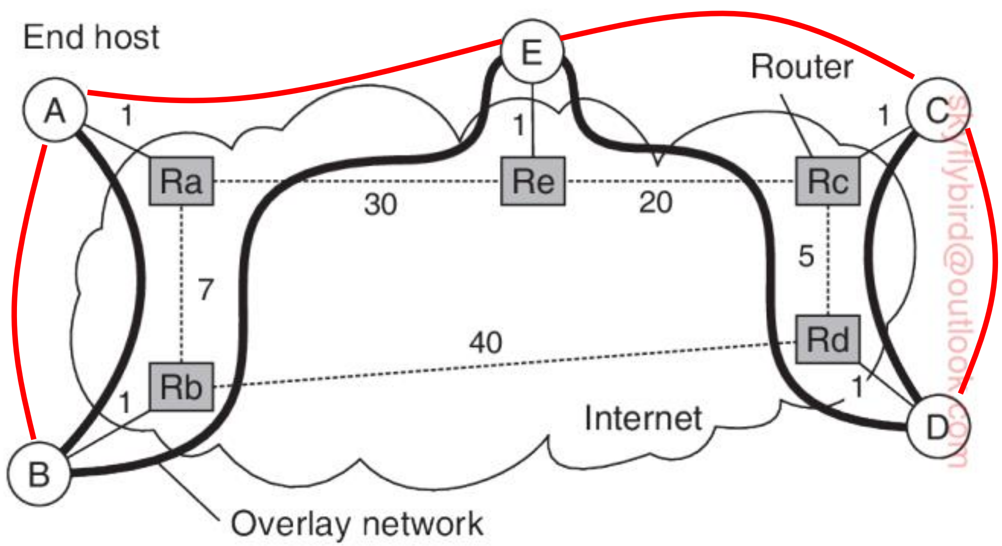
1. Initiator generates a multicast identifier *mid*.
2. Lookup *succ*(*mid*), the node responsible for *mid*.
3. Request is routed to *succ*(*mid*), which will become the root.
4. If *P* wants to join, it sends a join request to the root.
5. When request arrives at *Q*:
    - *Q* has not seen a join request before $\Rightarrow$ it becomes forwarder; *P* becomes child of *Q*. Join request continues to be forwarded.
    - *Q* knows about tree $\Rightarrow$ *P* becomes child of *Q*. No need to forward join request anymore.





Overlay的性能/质量

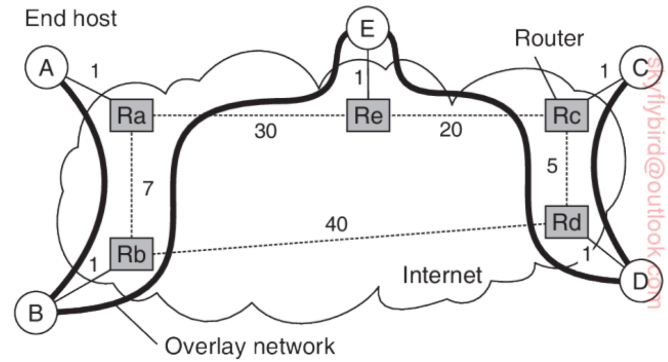
- Virtual link vs physical link
- Black tree: A-B-E-D-C, physical links?
- If change to red tree?



skyflybird@outlook.com

Figure 4.35: The relation between links in an overlay and actual network-level routes.

Overlay的性能/质量



- 三个性能指标
- Link stress:
  - counts how often a packet crosses the same link
- Stretch (relative delay penalty):
  - the ratio of the overlay network delay between two nodes over that of the underlaying network
- Tree cost:
  - the aggregated link costs

Broadcast vs. Multicast

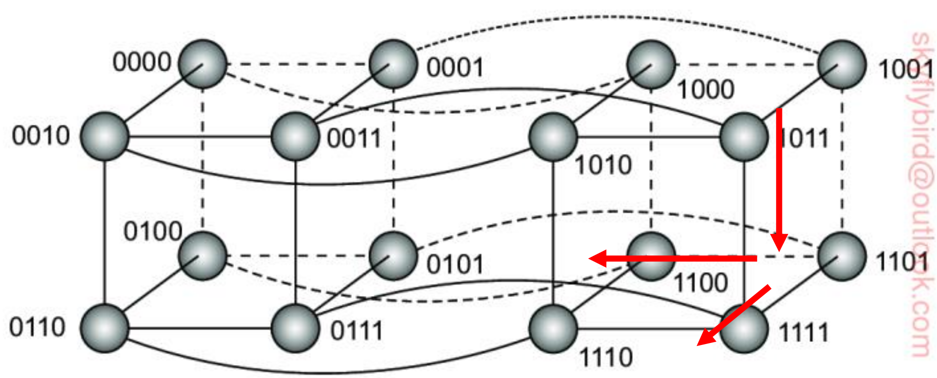
- 基于Overlay如何多播?
- 不同做法，各有利弊：
  - Overlay覆盖所有节点，用broadcast做multicast
  - Overlay覆盖目标组节点，部分中间节点做转发
  - Overlay只覆盖目标组节点
- Broadcast：发给所有节点
- Multicast：发给一组特定节点

消息的传播

- 最简单做法：flooding
  - Sending once per edge
- Cost
  - Unstructured overlay
    - 随机图: $M = \frac{1}{2} \cdot p_{edge} \cdot N \cdot (N-1)$
    - 概率转发：可以降低开销但是无法保证覆盖
  - Structured overlay:
    - Tree: N-1
    - Structured p2p: N-1

消息的传播

- Structured p2p: Hypercube
- Each edge is labeled with its dimension.
  - 0000->0001: 4, changing the 4th bit
- Initial node: 发消息给所有邻居
- Forwarders: 只转发给“高维度”边



初始1001节点

- (m,1) to 0001
- (m,2) to 1101
- (m,3) to 1011
- (m,4) to 1000

Figure 4.37: A simple peer-to-peer system organized as a four-dimensional hypercube.

消息的传播

- Structured p2p: Chord
- Divide nodes according to key values and neighbors
  - recursively
- E.g., initial node 9:
  - 28: $28 \leq k < 9$
  - 18: $18 \leq k < 28$
  - 14: $14 \leq k < 18$
  - 11: $11 \leq k < 14$
- Total cost
  - N-1

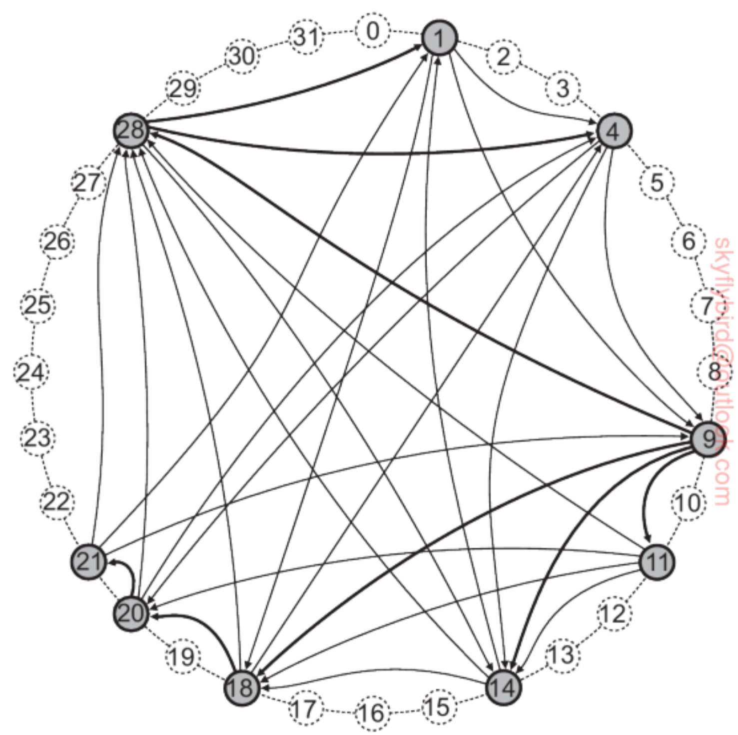


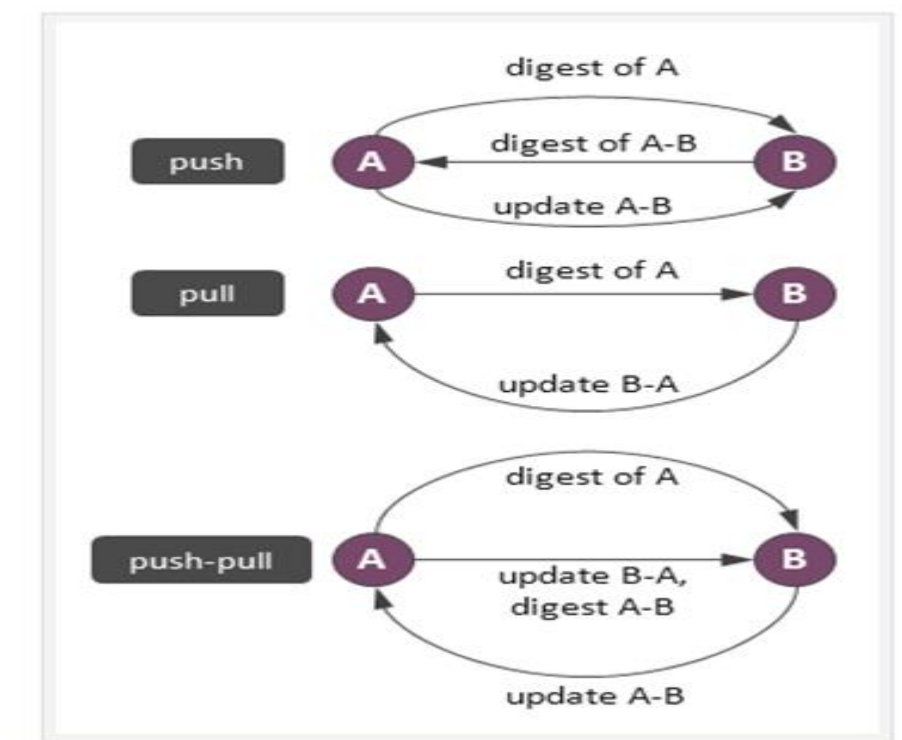
Figure 4.38: A Chord ring in which node 9 broadcasts a message.

Gossip-based Multicast

- Data dissemination with local information only
- Propagation in a gossip way
  - Each node randomly choose destinations
- Simple but no guarantee of delivery
  - Only guarantee of "eventual delivery"
- Especially suitable for many-to-many

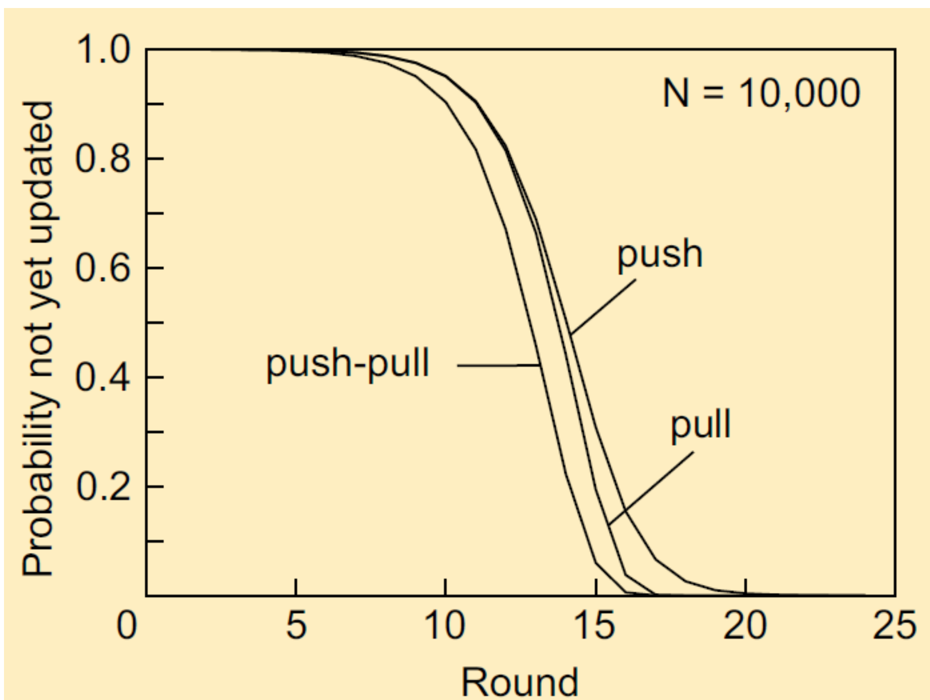
Gossip-based Multicast

- Three paradigms
  - P only pushes to Q
  - P only pulls from Q
  - P and Q send message to each other



Information Dissemination Models

- Anti-entropy propagation model
  - Node P picks another node Q at random
  - Subsequently exchanges updates with Q
- Eventual consistency



Information Dissemination Models

- Rumor spreading model
  - 节点P收到数据x后，push给随机选的邻居Q
  - 如果Q已经有x，P按照概率 p_{stop}停止传播

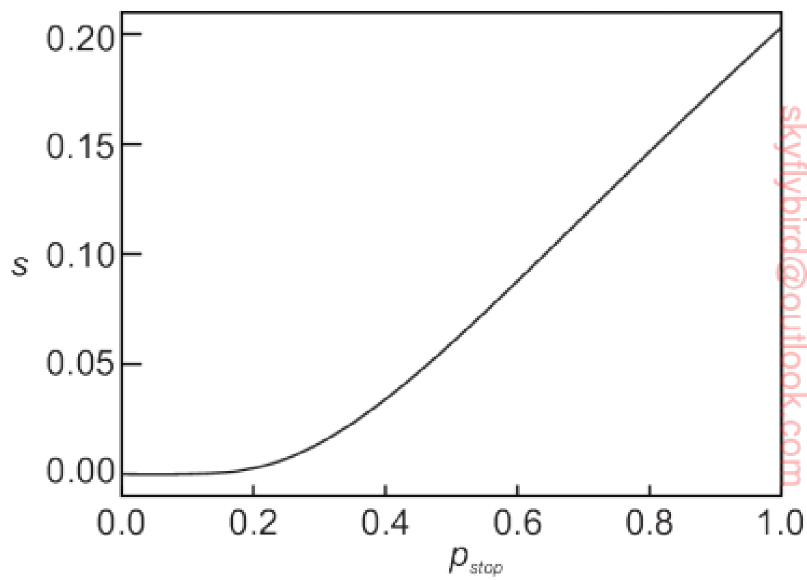


Figure 4.40: The relation between the fraction s of update-ignorant nodes and the probability p_{stop} that a node will stop gossiping once it contacts a node that has already been updated.

Comparisons of IPC Methods

- RPC:
  - Transparent
  - Synchronous nature
- Messaging
  - Socket: transient
    - General purpose
    - Simple, no powerful interfaces
  - MPI: transient
    - Parallel computing, mainly
    - Powerful with proprietary communication libraries, not robust
  - Message queuing: persistent
    - Asynchronous nature
- Streaming
  - For continuous data
- Multicasting
  - Overlay-base (established paths)
  - Gossip-based: probabilistic

Homework Questions

1. 请分析讨论RPC与一般的消息通信的关系、异同。
2. Gossip的多播与基于Overlay的多播各适于什么样的场景？请举例说明。