SCS 303: Distributed Systems.

3. Communication

IPC

- Interprocess communication is at the heart of all distributed systems.
- IPC is concerned with how processes on different machines can exchange information.
- Communication in distributed systems is always based on message passing as offered by the underlying network.
- Is the way of communication btn interprocesses of the sender and receivers machine, interms of sending and receiving explicit messages between them

IPC

- Development of large-scale distributed applications is extremely difficult if primitive communication facilities of computer networks are the ones in use
- Communication models
  - RPC (Remote Procedure Call)
  - MOM (Message-Oriented Middleware)
  - Data streaming

Types of Communication

- **Persistent**
  - An electronic mail system is a typical example of communication persistent communication
  - A message that has been submitted for transmission is stored by the communication **middleware as long as it takes to deliver it to the receiver.**
  - Middleware will store the message at one or several of the storage facilities
  - Not necessary for the sending application to continue execution after submitting the message
  - Receiving application need not be executing when the message is submitted.

Types of Communication

- **Transient**
  - A message is stored by the communication system only as long as the sending and receiving application are executing/running
  - The message will be discarded if the communication server cannot deliver the message to the destination server
- The communication system consists of traditional store-and-forward routers.
- If a router cannot deliver a message to the next one or the destination host, it will drop the message.

Types of Communication

- **Synchronous communication** - the sender is blocked until its message is stored in local buffer at the receiver end or the message has been delivered.
- **Asynchronous communication** - a sender continues immediately after it has submitted its message for transmission.
- This means that the message is (temporarily) stored immediately at the local buffer/the middleware upon submission.

Types of Communication

- Discrete
  - The parties communicate by messages, each message forming a complete unit of information
- Streaming
  - Involves sending multiple messages, one after the other, where the messages are related to each other by the order they are sent, or because there is a temporal relationship

RPC

- Many distributed systems have been based on explicit message exchange between processes.
- However, the procedures send and receive do not conceal communication at all, which is important to achieve access transparency in distributed
- Solution: Birrell and Nelson proposed allowing programs to call procedures located on other machines.

RPC

- When a process on machine A calls' a procedure on machine B, the calling process on A is suspended, and execution of the called procedure takes place on B.
- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing at all is visible to the programmer

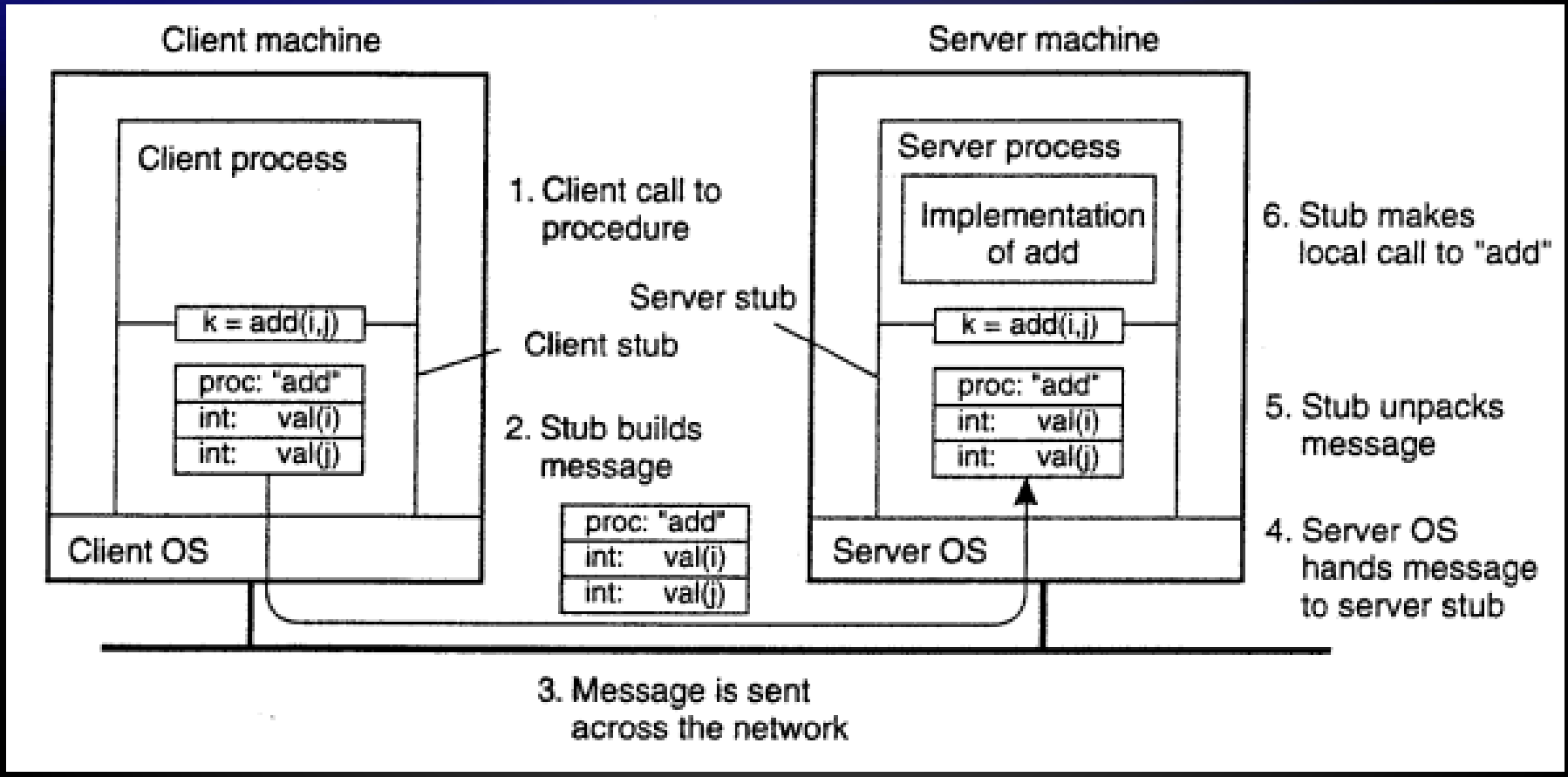
RPC Steps

- The client procedure calls the client stub in the normal way.
- The client stub builds a message and calls the local operating system.
- The client's OS sends the message to the remote OS.
- The remote OS gives the message to the server stub.
- The server stub unpacks the parameters and calls the server.

RPC Steps

- The server does the work and returns the result to the stub.
- The server stub packs it in a message and calls its local OS.
- The server's OS sends the message to the client's OS.
- The client's OS gives the message to the client stub.
- The stub unpacks the result and returns to the client.

RPC


Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
proc: "add"
int: val(i)
int: val(j)
Client OS
Server OS
1. Client call to procedure
Client stub
Server stub
2. Stub builds message
proc: "add"
int: val(i)
int: val(j)
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"

Asynchronous RPC

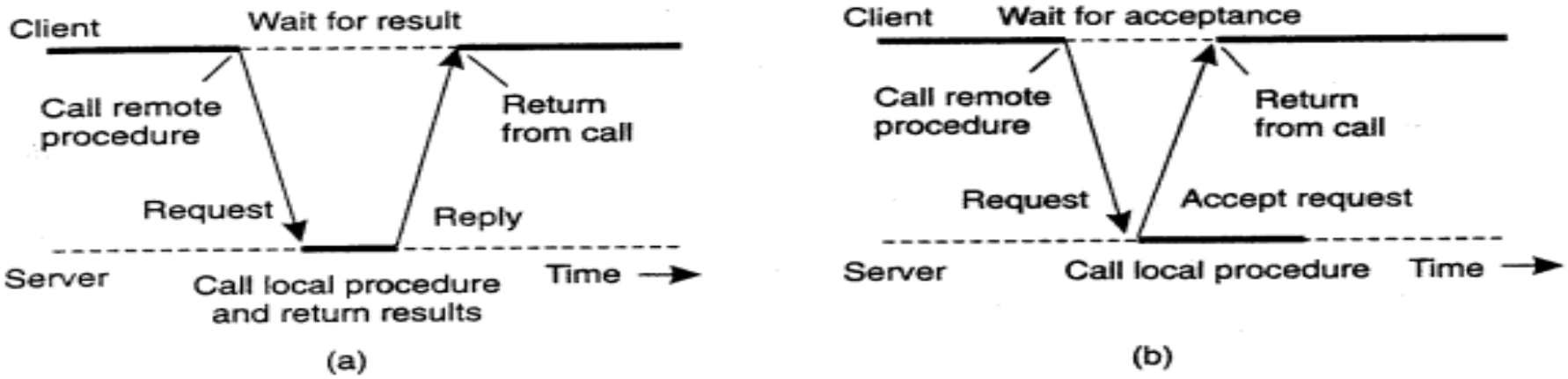


Figure 4-10. (a) The interaction between client and server in a traditional RPC. (b) The interaction using asynchronous RPC.

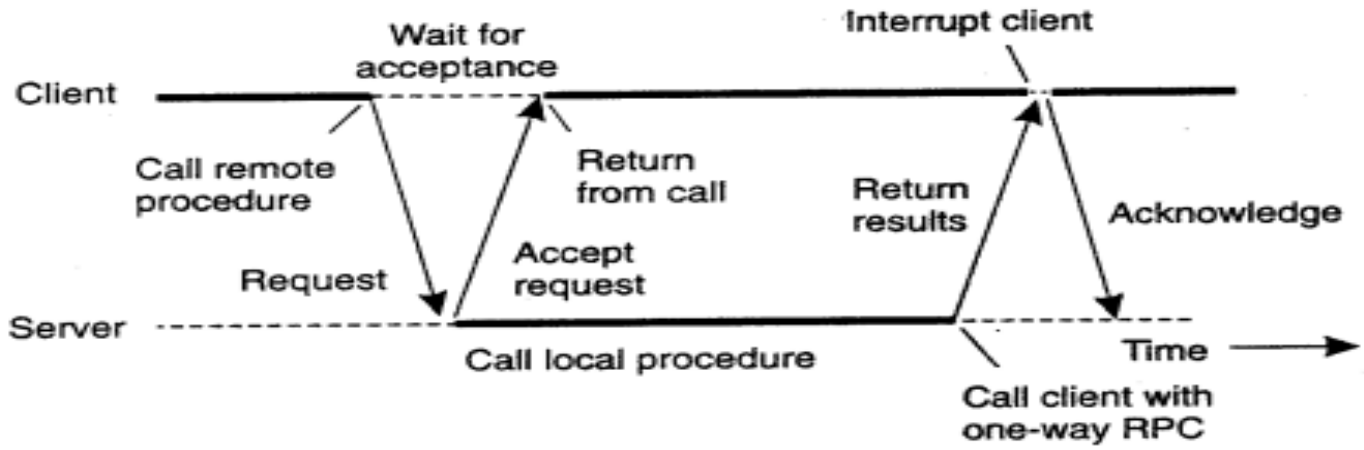


Figure 4-11. A client and server interacting through two asynchronous RPCs.

Message-oriented Communication

- Remote procedure calls and remote object invocations contribute to hiding communication in distributed systems, that is, they enhance access transparency
- Not suitable when it cannot be assumed that the receiving side is executing at the time a request is issued
- The inherent synchronous nature of RPCs, by which a client is blocked until its request has been processed, sometimes needs to be replaced by something else

Berkeley Sockets

- To appreciate Message Queue (MQ) systems, we look at sockets and MPI (Message Passing Interface).
- Sockets are used in RPC communication
- A socket is a communication end point to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read
- A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol.

Berkeley Sockets

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

Figure 4-14. The socket primitives for TCPIIP.

The Message-Passing Interface (MPI)

- With the advent of high-performance multicomputers, developers have been looking for message-oriented primitives that would allow them to easily write highly efficient applications with minimal overheads

MPI

- Disadvantages of Sockets
  - Are at the wrong level of abstraction by supporting only simple send and receive primitives
  - Were designed to communicate across networks using general-purpose protocol stacks such as TCP/IP hence are not considered suitable for the proprietary protocols developed for high-speed interconnection networks, such as those used in high-performance server clusters.

MPI

- Proprietary protocols developed for high-speed interconnection networks require an 'interface that can handle more advanced features, such as different forms of buffering and synchronization
- This resulted to interconnection networks and high-performance multicomputers that were shipped with proprietary communication libraries.
- All libraries were mutually incompatible which let to portability problems

MPI

- The need to be hardware and platform independent eventually led to the definition of a standard for message passing, called the Message-Passing Interface or MPI.
- MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users.
- MPI is designed for parallel applications and as such is tailored to transient communication.
- It makes direct use of the underlying network.

Message-Oriented Persistent Communication

- Message-queuing systems, or Message-Oriented Middleware (MOM provide extensive support for persistent asynchronous communication.
- They offer intermediate-term storage capacity for messages, without requiring either the sender or receiver to be active during message transmission

MOM

- An important difference with Berkeley sockets and MPI is that message-queuing systems are typically targeted to support message transfers that are allowed to take minutes instead of seconds or milliseconds.
- Basic idea behind a message-queuing system is that applications communicate by inserting messages in specific queues.
- These messages are forwarded over a series of communication servers and are eventually delivered to the destination, even if it was down when the message was sent

MOM

- Each application can have its own queue but in some cases applications may share a single queue.
- No guarantees are given about when, or even if the message will actually be read, which is completely determined by the behavior of the recipient
- Messages must be properly addressed

MOM

- Queues are managed by queue managers.
- Normally, a queue manager interacts directly with the application that is sending or receiving a message.
- There are also special queue managers that operate as routers: they forward incoming messages to other queue managers.
- In this way, a message-queuing system may gradually grow into a complete, application-level, overlay network, on top of an existing computer network.

MOM: Message Brokers

- An important application area of message-queuing systems is integrating existing and new applications into a single, coherent distributed information system.
- Integration requires that applications can understand the messages they receive.
- Same format?
  - The problem with this approach is that each time an application is added to the system that requires a separate message format, each potential receiver will have to be adjusted in order to produce that format

MOM: Message Brokers

- Better approach is to learn to live with different formats, and try to provide the means to make conversions as simple as possible.
- In message-queuing systems, conversions are handled by special nodes in a queuing network, known as message brokers
  - A message broker acts as an application-level gateway in a message-queuing system.
  - Its main purpose is to convert incoming messages so that they can be understood by the destination application.

Overlay networks

- An important part of managing MQ systems is connecting the various queue managers into a consistent overlay network.
  - An overlay network is a virtual network consisting of nodes and virtual links, which sits on top of an underlying network (such as an IP network) and offers something that is not otherwise provided

Overlay networks

- For example:
  - a service that is tailored towards the needs of a class of application or a particular higher-level service – for example, multimedia content distribution;
  - more efficient operation in a given networked environment – for example routing in an ad hoc network;
  - an additional feature – for example, multicast or secure communication.

Overlay networks

- Overlays are layers, but layers that exist outside the standard architecture (such as the TCP/IP stack) and exploit the resultant degrees of freedom. E.g. modes of addressing(routing based on hash tables), protocols used and routing approach

Stream-oriented Communication

- Communication as discussed so far has concentrated on exchanging more or less independent and complete units of information.
- The characteristic feature of this type of communication is that it does not matter at what particular point in time communication takes place.
- Although a system may perform too slow or too fast, timing has no effect on correctness

Stream-oriented Communication

- There are also forms of communication in which timing plays a crucial role.
- Assume that the audio stream represents CD quality, meaning that the original sound wave has been sampled at a frequency of 44, 100Hz.
- To reproduce the original sound, it is essential that the samples in the audio stream are played out in the order they appear in the stream, but also at intervals of exactly 1/44, 100 sec.
- Playing out at a different rate will produce an incorrect version of the original sound

Stream-oriented Communication

- To capture the exchange of time-dependent information, distributed systems generally provide support for data streams.
- A data stream is a sequence of data units.
- Discrete e.g. TCP/IP connections are typical examples of(byte-oriented) discrete data streams
- Continuous e.g. Playing an audio file typically requires setting up a continuous data stream between the file and the audio device.

Transmission Modes

- Timing is crucial to continuous data streams.
- To capture timing aspects, a distinction is often made between different transmission modes.
- Asynchronous: data items in a stream are transmitted one after the other, but there are no further timing constraints on when transmission of items should take place e.g. in transfer of a file

Transmission Modes

- Synchronous: there is a maximum end-to-end delay defined for each unit in a data stream
  - Whether a data unit is transferred much faster than the maximum tolerated delay is not important
- Isochronous: it is necessary that data units are transferred on time, data transfer is subject to a maximum and minimum end-to-end delay, also referred to as bounded (delay) jitter.

Streams and Quality of Service

- QoS requirements describe what is needed from the underlying distributed system and network to ensure that, for example, the temporal relationships in a stream can be preserved.
- QoS for continuous data streams mainly concerns timeliness, volume, and reliability.

Streams and Quality of Service

- Properties that should be defined
  1. The required bit rate at which data should be transported.
  2. The maximum delay until a session has been set up (i.e., when an application can start sending data).
  3. The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient).
  4. The maximum delay variance, or jitter.
  5. The maximum round-trip delay

Enforcing QoS

- Since the underlying system offers only a best-effort delivery service, a distributed system can try to conceal as much as possible of the lack of quality of service.
- Internet provides a means of differentiating classes of data by means of its differentiated services
- Outgoing packets can be marked as
  1. expedited forwarding class: Essentially specifies that a packet should be forwarded by the current router with absolute priority

Enforcing QoS

2. Assured forwarding class: Defines a range of priorities that can be assigned to packets, and as such allows applications to differentiate time-sensitive packets from noncritical ones
   - Also defines ways to drop packets if the network gets congested.

- A distributed system can also help in getting data across to receivers by using buffers to reduce jitter

Enforcing QoS

- Assuming that packets are delayed with a certain variance when transmitted over the network, the receiver simply stores them in a buffer for a maximum amount of time.
- This will allow the receiver to pass packets to the application at a regular rate

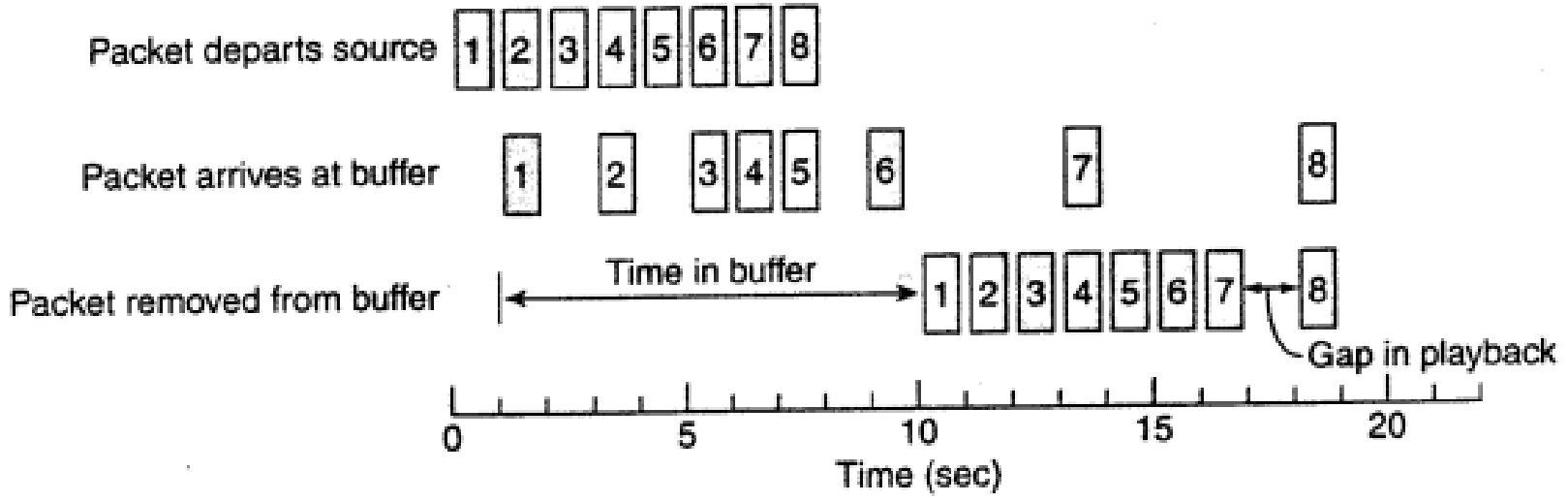


Figure 4-27. Using a buffer to reduce jitter.

Enforcing QoS

4. Since retransmission is not an option loss of packets is handle using FEC
   - Forward Error Correction(FEC:): carefully designed redundancy that allows the receiver to detect and correct a limited number of errors occurring anywhere in the message without the need to ask the sender for additional data
   - k out of n received packets is enough to reconstruct k correct packets

Multicast Communication

Multicast Communication

- Allows sending of data to multiple receivers
- Can be provided in two ways
  1. Application-Level Multicasting
  2. Gossip-Based Data Dissemination

Multicast Communication

- Application-Level Multicasting
  - The basic idea in application-level multicasting is that nodes organize into an overlay network, which is then used to disseminate information to its members.
  - Overlay Network design
    - Tree – overlay path between every pair of nodes
    - Mesh - every node will have multiple neighbors and there exists multiple paths between every pair of nodes.

Gossip-Based Data Dissemination

- Disseminating of information relies on epidemic behavior
- The main goal of these epidemic protocols is to rapidly propagate information among a large collection of nodes using only local information.
  - Event information
  - Background data dissemination
- Good for sending data in unreliable networks

Gossip-Based Data Dissemination

- Periodically, a node picks another at random and gossips with it.
- After each round, nodes with the information double (approximately)
- Exponential (follows logarithmic time in network size)
- Robust, even if some messages get lost
- Analogy - Internet routing

Gossip-Based Data Dissemination

- Definitions from epidemics
  - Infected
    - A node that has received an update
  - Susceptible
    - One that is yet to receive an update
  - Removed
    - An updated node that is not willing or able to spread its data

Information Dissemination Models

Anti-entropy propagation model

- In this model, a node P picks another node Q at random, and exchanges messages with Q.
- There are three approaches to exchanging updates:
  1. P only pushes its own updates to Q
     - If P is more updated, push changes to Q
     - if many nodes are infected, the probability of each one selecting a susceptible node is relatively small

Information Dissemination Models

2. P only pulls in new updates from Q
   - Ask remote copy if more recent, then pull changes
   - spreading updates is essentially triggered by susceptible nodes.
   - If many nodes are infected, chances are large that a node(susceptible) will contact an infected one to subsequently pull in the updates and become infected as well
3. P and Q send updates to each other (i.e., a push-pull approach)

Information Dissemination Models

Rumor Mongering

- Gossip a variant of anti-entropy
- If site realizes rumor has spread sufficiently, it stops sharing.
- P may lose interest in spreading the update any further with
  - Feedback-based probability (Stop with probability 1/k if site was already infected)
  - Blind probability (always stop with probability 1/k)
  - Fixed probability (stop after k nodes report they are already infected)

Gossip-Based Data Dissemination

- Gossiping turns out to be an excellent way of rapidly spreading news.
- However, it cannot guarantee that all nodes will actually be updated
- Algorithms for sorting nodes by id, arranging nodes in a tree, searching information, getting aggregates

Gossip-Based Data Dissemination

- Removing Data (Deletion)
- Side-effect of epidemic algorithms: spreading the deletion of a data item is hard.
- The essence of the problem lies in the fact that deletion of a data item destroys all information on that item
- Consequently, that node will eventually receive old copies of the data item and interpret those as updates on something it did not have before

Information Dissemination Models

- Solution: record the deletion of a data item as just another update, and keep a record of that deletion.
- Deletions should eventually be cleaned up.