CMPSCI 677 Operating Systems **Spring 2014**

Lecture 10: February 28

Lecturer: Prashant Shenoy *Scribe: Nimish Gupta*

10.1 Communication

Assume there are two end points that want to communicate with each other: sender and receiver. A message goes through series of nodes (routers) after leaving sender and before reaching receiver. These routers could be typical network routers or nodes in an overlay network. Messages in a communication system can be classified along dimensions of persistence and synchronocity

- **Persistent**
  Persistence means that the network is capable of storing messages on a hop for an arbitrary period of time until the next hop becomes ready to take delivery of the message. Email and ground deliveries are good examples of persistent communication.
- **Transient**
  Messages are always forwarded to next hop assuming it is always available. If the next hop is not available then the message gets discarded. For example, Transport-level communication discards the message if the process crashes for any reason. The system will not store messages.
- **Asynchronous**
  Asynchronous communication means that the sender does not block for receiver to reply. It continues immediately after submitting the message.
- **Synchronous**
  A process doing synchronous communication blocks until an acknowledgement is received for the sent message. Depending on type of acknowledgement required, synchronous communication can be classified further.

10.1.1 Persistent Asynchronous Communication

When sender sends a message, the receiver need not be running. The message is stored somewhere along the path. Message is delivered to receiver when it comes online. When the reciever comes online and receives the message, it is not required that the sender be running at that time. Example: email.

10.1.2 Persistent Synchronous Communication

The sender blocks immediately after sending the message, waiting for an acknowledgement to come back. The message is stored in a local buffer, waiting for the receiver to run and receive the message. Some instant message applications, such as Blackberry messenger, are good examples. When you send out a message, the app shows you the message is "delivered" but not "read". After the message is read, you will receive another acknowledgement.

10.1.3 Transient Asynchronous communication

Since the message is transient, both entities have to be running. The sender doesn't wait for response because it is asynchronous. UDP is an example of this type.

10.1.4 Receipt-based transient synchronous communication

The acknowledgement sent back from the receiver indicates that the message has been received by the OS at the other end. The acknowledgement does not say anything about whether the application at other end has received the message and has started processing it.

10.1.5 Delivery-based transient synchronous communication

The acknowledgement comes back to the sender when the application other end actually have received the message. The acknowledgement does not mean that the other end has started processing the message. Asynchronous RPC is an example.

10.1.6 Response-based transient synchronous communication

Acknowledgement signifies that the message was processed by the remote application and the acknowledgement could very well be the result of the processing. Example : RPC.

Sockets are example of transient communication. TCP/IP acknowledgements signifies that the OS at other end has received the message. Various sending primitives of MPI (in last lecture) cleanly maps to receipt-based, delivery based and response based modes of communication. The mode of communication to use is determined by the semantics of the application.

10.2 Message-oriented Persistent Communication

This section explains what kind of middleware is required to perform persistent communication

10.2.1 Message Queuing System (MQS)

A MQS supports asynchronous persistent communication. A persistent queue exists between the sender and the receiver and stores messages when sender/receivers are inactive. It takes messages from the sender and passes it down to the receiver whenever it is ready. It's a queue that is stored on disks, so you can store the message for an arbitrary amount of time. There are more than one persistent queue between the end points. However, there is no guarantee that messages will be read. Example application: email

There are four abstract methods needed to implement a message queuing system.

- **put**: append message to the queue

- **get**: get the message from the queue
- **poll**: poll to see if the queue is empty or not
- **notify**: notify the sender when the message is put into the queue

In a more general MQS architecture, there are usually many queues in the middle. There are application-level routers that implements the message queuing system. Messages will be delivered hop by hop, and they will be queued until the next receiver is ready.
There may also be nodes called message broker, providing a higher level service to applications. Applications can implement a pub-sub paradigm with the help of broker nodes. Whenever application wants to send a message, it does not need to specify recipients address but only the type of message it is sending and all nodes that are interested (subscribers) in those types of messages shall get notified. Message broker maintains the information of nodes that have registered for messages and delivers a message whenever it becomes available. Example : email mailing list.

MQS Case study : IBM WebSphere MQ

Queue is called a message channel in this system. This is a middleware which is composed of queue managers, which manages the queue, and a channel agent, which manages the packet transportation. There are also other open source MQSs.

If disks that act as persistent queues only have finite capacity, there should be a policy deciding how long are the messages queued. Email, for example, queues the mail for a certain amount of time before the receiver is available.

10.3 Message Queueing Model

4 flavours of how communication can take place

- Both the sender and receiver are active at the same time
- Sender is running but the receiver is not running. The message needs to be queued to be delivered to sender when it comes back online
- When receiver takes delivery of the message, the sender is inactive
- Neither of both sender and receiver are running but there is a message in transit.

10.4 Stream-Oriented Communication

Audio and video streaming are good examples of stream-oriented communication. Server sends the video file in smaller pieces, and it has to send them continually in time. There are timing constraints to assure good performance, which do not exist in message-oriented communication. Late data are not of much use. If data does not arrive in time then one can observe playback glitch.
An important characteristic of stream oriented communication is isochronous communication. There are timeliness constraints in stream-oriented communication. The network has to deliver data on time, or the users don't get satisfactory performance. There is some timing information embedded in how data is sent,

received and played out. Example: To play a decent quality video, client needs to play at a minimum frame rate of 30 fps (HD video can generally go upto 60 fps). Another characteristics is that this type of communication is server-push. The client initiates the transfer through a request to server. After that, there are no explicit request for data from the user. The streaming server continues to send data to the client, and the client simply keeps listening on the socket and receiving data. There are also client-pull streaming systems as well.

There are two classes of streaming depending on what type of data: stored data and live. Google hangout is an example of live streaming, and youtube is an example of stored data streaming. Live streaming has even more stringent constraints, because people have lower tolerance of lag in live streaming. There could be multiple receivers for one source, which can be done by a mechanism called multicast. Live stream of a sporting event is an example of multicast.

10.5 Streams and Quality of Service

The application is demanding a certain level of quality. The network and end system should meet these some requirements or the quality of service will not be satisfied. Bandwidth, delay, and lost constraints are examples of possible demands. Early streaming system are built on UDP because data retransmission of TCP causes more problem in streaming (Retransmission of TCP is not of much help in case of lost data as late data is as good as no data in streaming systems). Quality of video is another requirement of quality of service. For example, on youtube allows you could set the playback quality of the video. Moreover, It can estimate online the quality of packet streaming. If the system sees a lot of lost packets, it will lower the quality of the video. The system could have requirements on jitters, which is the variance of the delay. Bandwidth could be either fixed bandwidth or variant bandwidth. There are two kinds of encoding scheme, fixed rate encoding and variable rate encoding, in which the encoding rate could vary depending on the encoded data. In the old days, People assume that when the server starts streaming, it would tell the network the requirements it needs to have a satisfactory streaming, then the network would reserve resources for this stream. This is not implemented nowadays because it is difficult for the network to guarantee to provide resources. However, QOS is still needed.

10.5.1 token bucket

One mechanism to enforce constraint on bandwidth is the token bucket, or leaky bucket. It is also called the policing mechanism. Token bucket is an OS level mechanism. You could attach it to applications or a socket end points to enforce a constraint on the end point. You specify two parameters for the bucket: rate r and burst b, and this mechanism guarantees that the transmission rate does not exceed r. Once in a while, you can send a burst b of packets. The amount of data you can send at a time t is bounded by the equation: r*t+b. When you start the token bucket system, there are b tokens in the bucket. Every time an application generates a packet, it has to grab a token before it enters the network. Every second you generate r new tokens and add them into the bucket. The bucket can hold a maximum of b tokens. If the bucket is empty, no packets could pass. If an application requires rate r and burst b from the network, this mechanism restricts the application to meet the requirements it specified itself. This technique is used for mutiplexing network resources among VMs.

10.5.2 Receiving end

At the receiving end, packets arrive with jitters . There may be large time interval between each packet arrival. You won't play the video when the first packet arrives. Instead, you buffer some number of frames before you start playing. You will have enough packets to play even though packets arrive at different speed, so long as the buffer doesn't empty up. If packets arrive too slow and the buffer underflows, you will see a message saying that the buffer is "rebuffering". The system is waiting for the buffer to fill up before the video plays again. In this case, you should increase the buffer length to insure that the buffer never empties. The video player, not the network, is responsible for guessing the appropriate buffer length. The user has to wait longer if the buffer is larger, because you have to wait for the buffer to fill up. One the other hand, there will be glitches if the buffer is too short. The player may guess a certain length, and adjust it as the video is streaming.

10.5.3 Dealing with lost packets

Assume that one packet contains four frames (which is actually impractical because in reality one frame doesn't fit into one packet). Now during transmission, the third packet, which contains frame 9,10,11,and 12, gets lost. The client experience a glitch because of the missing frames in the third packet. When the video resumes, it skips a certain part of the video because frames containing that part disappeared. Another way of transmission is to scramble the frames. You take 16 frames, for example, then scramble them into the first four packets. The third packet now contains frame 3,7,11,and 15. If the third packets gets lost again, although you lose the same amount of frames, by spreading the losses, user sees better performance, because tiny glitches are less perceptible than large glitches. This mechanism assumes that the receiver end can reorder the packets, which is not a problem because frames are numbered. The bigger issue is that this mechanism also requires a larger buffer. You have to wait for all 16 frames to arrive before you can play the video.

10.5.4 HTTP Streaming

Entire video file is partitioned into smaller chunks of data. Each chunk has some number of frames in it. Video file is encoded and stored at different bitrates: low, medium, high for example, to support different clients over different capacity networks. When video is initially requested, the server start with medium bitrate and asks the client if there are any reception problems. If there are not any reception problems then server assumes a thicker network pipe and sends the next chunk with higher bitrate. If the client reports a problem then server lowers the quality of the next chunk of video. A difference is that this streaming is client driven due to the fact that HTTP is a client driven protocol. This technique is called Direct Adaptive Streaming over HTTP (DASH). Youtube and netflix work this way.