CMPSCI 677 Operating Systems **Spring 2013**

Lecture 10: February 25

Lecturer: Prashant Shenoy *Scribe: Timothy Wang*

10.1 Communication

Assume there are two end points that want to communicate with each other. There are multiple intermediaries, such as nodes and routers. Also, there are OSs that implement the TCP/IP. There are a few characteristics of a communication.

- **Persistence:**
  Persistence means that the network is capable of storing messages for a arbitrary period of time until the next receiver is ready. Email and ground deliveries are good examples.

- **Transient:**
  Message is stored only so long as the next receiver is ready. For example, Transport-level communication discards the message if the process crashes for any reason. The system will not store the message.

- **Asynchronous**
  Asynchronous communication means that the sender is doing nonblocking sending. It continues immediately after submitting the message.

- **Synchronous**
  Synchronous communication blocks the process until the message is received or the sender gets a response from the server.

10.1.1 persistent synchronous communication

The sender is blocked when it sends the message, waiting for a 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.

10.1.2 Transient asynchronous communication

Since the message is transient, both entities have to be running. Also, the sender doesn't wait for responses because it is asynchronous. UDP is an example.

10.1.3 Receipt-based transient synchronous communication

The acknowledgement sent back from the receiver indicates that the message has been received by the other end. The receiver might be working on some other process.

10.1.4 Delivery-based transient synchronous communication

The acknowledgement comes back to the sender when the other end actually takes control of the message. Asynchronous RPC is an example.

10.1.5 Response-based transient synchronous communication

The sender blocks until the receiver processes the request and sends back a response. RPC is an example.

There is no clean mapping of TCP to any type of communication. From an application standpoint it maps to transient asynchronous communication. However, in a protocol standpoint, it maps to a receipt-based transient synchronous communication if it only has a one-size window.

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

The persistent queue exists between the sender and the receiver. 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.

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 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.

IBM WebSphere MQ is a example of message queuing system. This is a middleware which compose 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.

10.2.2 Message Brokers

It transforms messages to the form that is needed. It can also be used as a filter. This is used in pub-sub systems. The receiver could subscribe to a certain type of message, and when the message arrives at the broker, it filters the message interested to the subscriber.

10.3 Stream-Oriented Communication

Audio and video streaming are good examples of stream-oriented communication. You send the video file in smaller pieces, and you have 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 actually not much use. An important characteristic 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. Another characteristics is that this type of communication is server-push. 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. Skype 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.

10.4 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. Nowadays, modern systems use TCP protocol to stream. 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.

10.4.1 token bucket

One mechanism to enforce constraint on bandwidth is the token bucket, or leaking 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.

10.4.2 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.

10.4.3 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.

10.4.4 Forward error correction

In the case above, we assumed that there is not time to retransmit lost frames. Only when the buffer is large could you have time to retransmit data. You can use forward error correction(FEC) to reconstruct the lost data. What FEC does is that it sends redundant data. In the worst case it sends each packet twice, which requires double bandwidth. There are more efficient techniques that implements this without increasing the requirement of bandwidth.