CMPSCI 677 Operating Systems **Spring 2018**

Lecture 11: March 5

Lecturer: Prashant Shenoy *Scribe:* ***Akul Siddalingaswamy***

11.1 Enforcing QOS: Forward Error Connection (FEC)

Forward error correction as shown below ensures QOS by making sure there is no major delay in the frames while playing. This is achieved by scrambling the order in which the frames are sent. This makes sure that the data loss is distributed and imperceptible to the human eye or ear depending on whether its a video or audio file.

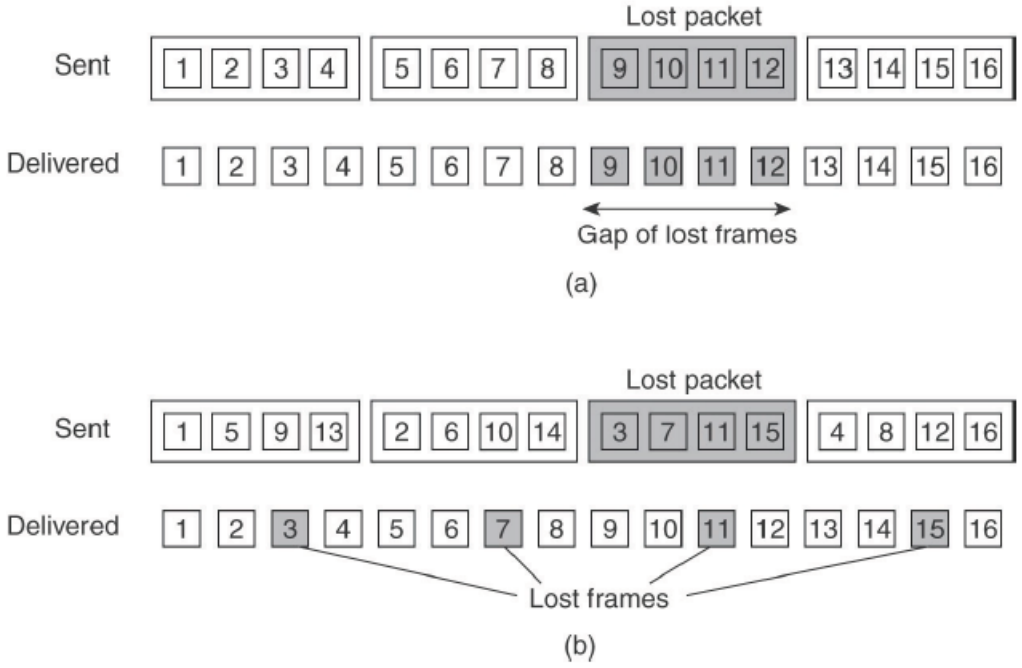

Figure 11.1: Forward Error correction

11.2 HTTP streaming

Last time we talked about audio and video streaming as a different way for how distributed communications between clients and servers. Streaming typically uses a server push architecture where we just press player and the server starts streaming data to you. However, most of the video player in the Internet today uses HTTP. HTTP is a protocol that was designed for requesting web pages and images and getting responses. It is not possible if you use HTTP because HTTP is fundamentally a client pull protocol. It was not originally designed for streaming; it was a request response protocol.

Nevertheless, it has been adapted to support audio/video streaming. What happens to HTTP streaming is; Firstly, the file is split into small chunks of files that has a few seconds of video each. When player send explicit request for each of these files. When one file is playing, you can request for other files in the future. Secondly, HTTP streaming allows to deal with lost and late by modifying the quality of video play back in real time. Each video file is encoded as different resolution and quality. When the content is being played, the player will look at how the network is behaving in any given time. If the data is not arrived in steady rate or some network problem is observed, client will automatically request the alternatives lower quality segments. If the network condition is good, client can request for better quality and so on.

This HTTP streaming protocol is actually referred to DASH (Direct Adaptive Streaming over HTTP) protocol. The way HTTP is used for streaming propose.

Another issue is, many early streaming protocol uses the UDP protocol. HTTP runs over TCP but you can get on TCP transmission taking longer time and so on just by improving buffering capability at the client. There is something to keep in mind: despite UDP being a better transport protocol for time and constraint and so on, by far all of the streaming protocol used in the Internet is HTTP streaming protocol.

11.3 Stream synchronization

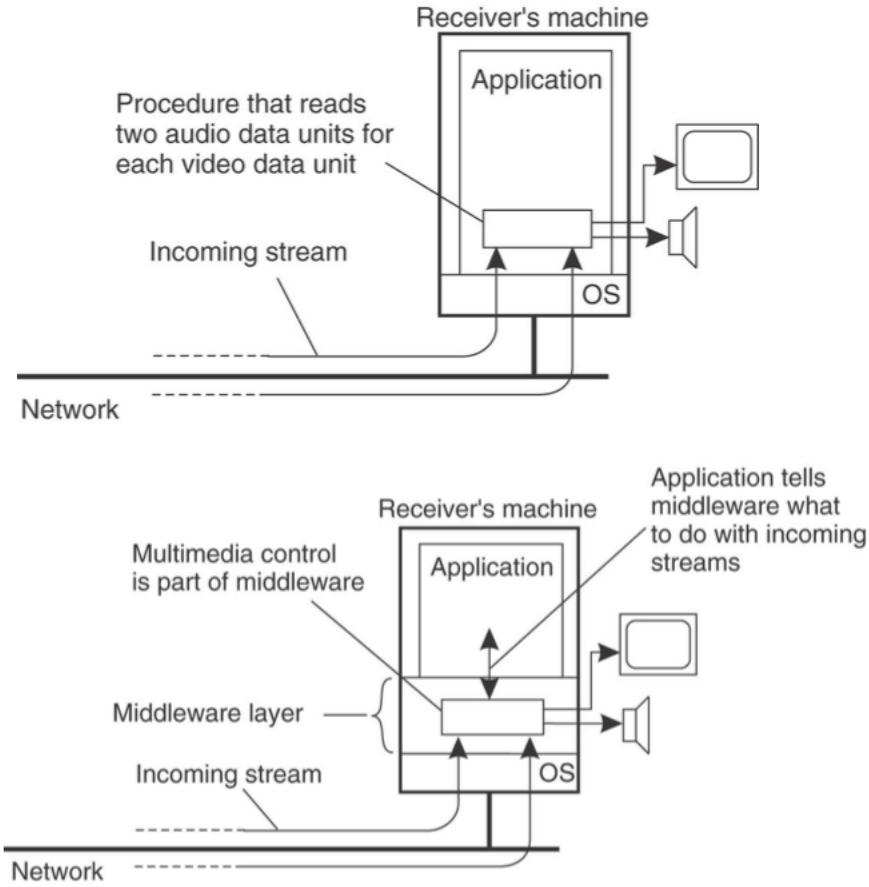

Figure 11.2: video by camera and audio by microphone

If you take any video stream, typically it will have multiple sub streams encoded in it. Every video file will at least have a video stream and an audio stream. The audio data goes with the video data. So it essentially has two stream and many times more than two stream. Even audio stream may be split. If it is a stereo audio, it will be split into multiple audio channels that can be played into left speaker and right speaker and so on.

When the data arrives at the machine, a machine should split the streams. So that different stream is played on different devices: video stream is played on the screen and audio stream is played on the speaker. Stream synchronization is used to ensure a good quality. When the file is being shown, the audio data played on speaker should goes with the video image. The lack of stream synchronization will cause the loss of lip-sync. Lip-sync means the audio played out the same time as users see the lip move.

But that is difficult for both the capture and the play back is happened on independent devices. Like what the below picture shows, video is captured and generated by the camera and audio by the microphone. So we will need a time stamp with the streams so as to know which audio sample and which video sample should be put together. Stream synchronization should be achieved by the time stamp carefully on each of the streams.

Other than the communication between two entities (the sender and the receiver), in many cases, there are also group communication. The data is send once and received by multiple receiver. In this case, what is used is multicasting protocol. This multicasting requires multicast distribution tree. The root of the tree is the sender, the data is send all through the tree and each of the leaves is the multicast receiver. Any multicast framework requires a construct of the multicast tree. Native IP multicast means multicast protocol supported at the IP level. The tree is handled by routers and so on. Application level multicast means overlay nodes that essentially construct the tree and the overlay down to the network. Just as P2P network is an overlay network where the P2P topology independent of the underlying network topology, an application multicasting tree is also an overlay network where the tree topology for multicast is independent of the underlying network topology. So peers in chat dont have just as the nodes in the network sense, because they are arbitrary nodes and overlay network.

What should be keep in mind is that this is a form of distributed communication where there is one sender and multiple receivers. And that is something that we might have to use in way of different context or the lab and so on

11.4 Naming/distributed naming

Essentially all resources in any distributed system or applications are given names. Other entities in your system can these resources using names. So there must be something that can map the resource to the underlying entity of the resource. To do this you are going to need a naming service.

Giving an example: if you use RPC or RMIs in the Lab, you have to register a server to a naming service. The server will have to have a name associated with it. When client need to bind to a server, it has to do the name look up. Basically the name lookup allows you to specify an addressed server by name, and it map to, in this case, an IP address and port number the server is listening.

If you have a large distributed system, the number of names in your distributed system will be large. So you are not going to have one name server that will handle all the naming request. The naming system itself has to be distributed in nature. We will use Domain Name System (DNS) as a distributed naming system example to see how to construct them. We will use machine name to IP address mapping as example to what kind of name that have to resolve. What you have to keep in mind is that you can have arbitrary name in any distributed system.

Lets see what will happen if you tap www.cnn.com: your browser will do name lookup in DNS. DNS will map that name to the IP address of the machine. User does not need to know any of them other than the name of the server which, in this case, is the URL. That is going to use name resolution.

11.4.1 Naming approaches

Here are two approaches for distributed naming. When you have a lot of entities that have name and now performing naming lookup, your distributed system is going to be distributed in nature. The way you are going to construct the distributed naming system is to one of the two mechanisms. Either you are going to use a hierarchical architecture where the name will be in hierarchical in nature and the naming system will be hierarchical. Or you are going to use a distributed Hash Table (or DHT) and a P2P-like name lookup to actually look up names, in which you are actually doing the key value look up. Key is the name and value is what you are lookup. Same concept can be used in naming system as well.

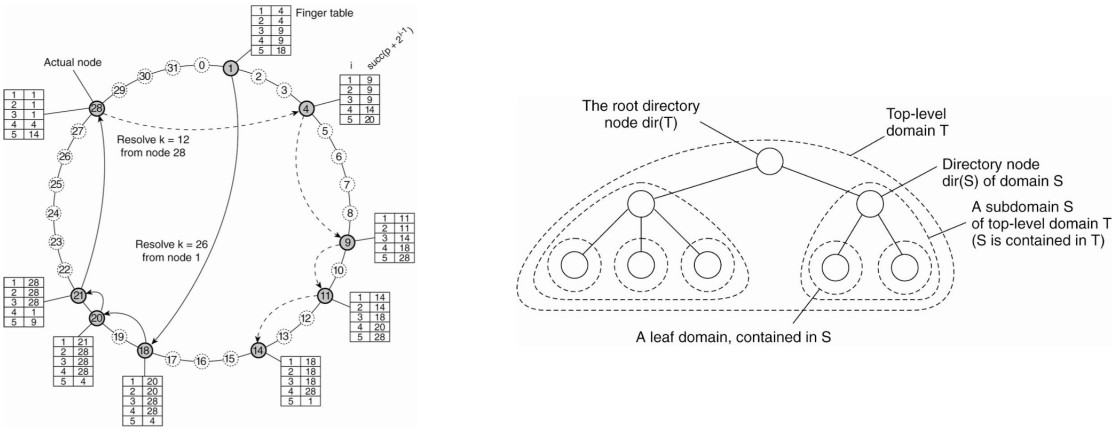

Figure 11.3: video by camera and audio by hierarchical and P2P-like

Lets take a file name as a simple example. When you open a file, you will specify a name in your file program. Underneath, your operating system will map it to an internal name. The file system does not actually use file name to organize files. It actually uses ids which are typically integers. So internally every filename has a unique id. So it is throughout naming based system to map filename of a file to underlying id. The file systems name space is typically hierarchical. For example, if you are going to find /home/steen/mbox, you will first look at the root directory / and fine the subdirectory home. Then you open the subdirectory home and then for the next level of subdirectory. So you are actually descending down the hierarchy literately. If you finally find the file, you found the internal id then you open the file. This is an example of how name resolution look like.

The same concept works for any other naming system. The name has hierarchical component and you are going to look at each of the component and then you trying to find the resource the user wants to find or the application is trying to address.

11.4.2 Resolving File name across machines

For distributed file system, some of the files are stored on the server and not on the clients, so name distribution has to become distributed.

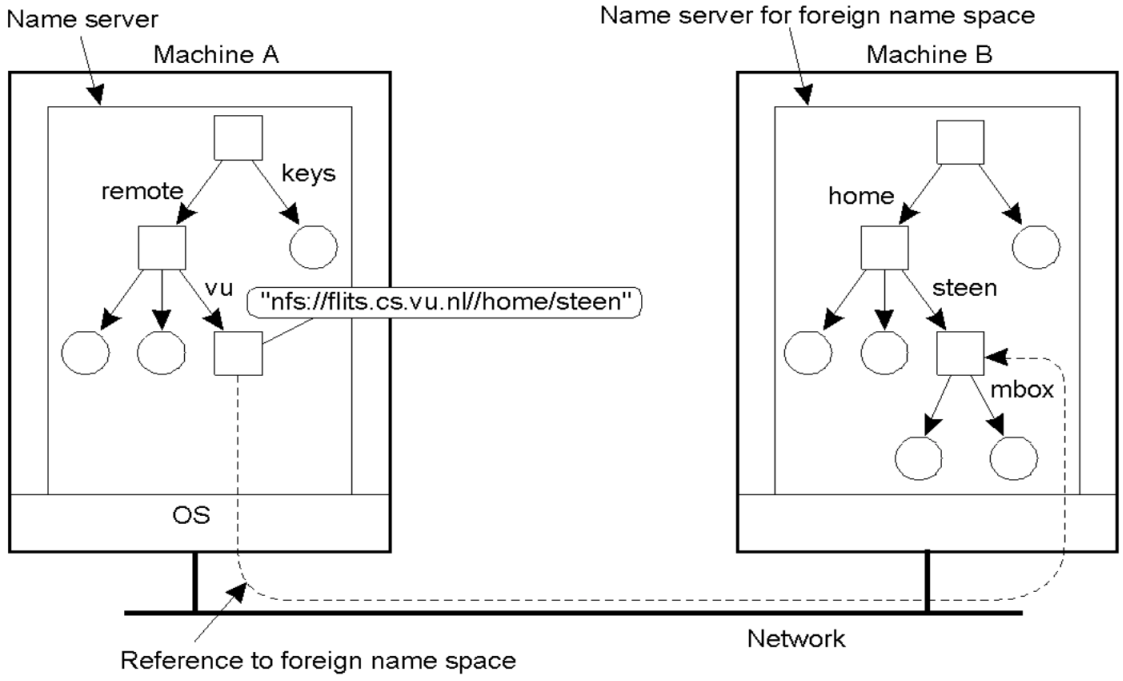

Figure 11.4: resolving file name across machines

Let take an example where you have machine A (which is a client) and machine B (which is a server). You have taken some file stored on the directory from the file server and you have mapped it on to the client using a mount protocol. Mount protocol simply establishes a mapping of a local directory to remote directory. So when you open the local directory, you are actually seeing files in remote directory as you have just constructed a mapping using the remote directory. So in this case what is going to happen is if you are now looking for a file, you are descending down the hierarches and you hit on the mount point where the OS know explicitly that everything underneath this directory is file in the remote directory. Then it says I did partial name resolution and now let me now send the request to the server and ask it to resolve where stored the name. Because the file actually stored on the sever. So you are going to look partial name resolution locally, figure out what you are trying to access of the remote file in the distributed file system, send the file name to the server and ask the server to give a handle on that. What we have generalized from the previous case is part of the name resolution is done at the client. Name resolution in this case was with multiply entities and now it become distributed.

11.4.3 Name space distribution

Naming in large distributed systems may be global in scope. If you look into Internet, naming system actually has millions of nodes. Each of them needs a naming service and names of themselves are part of the naming space. So naming system has to serve potentially millions of devices in very large distributed systems. So the naming system has to be highly distributed.

For internet, we use DNS to map name to IP address. DNS Name space is distributed and has three logical layers:

1. Global layer, highest level nodes and the root of hierarchy. Mostly static;
2. Administrational layer: the organization that are part of each of the domain;
3. Managerial layer: lowest layer, are actual nodes. Where there are frequent changes.

The below chart illustrates some of the requirement of some of the domains:

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

Figure 11.5: requirement of some of the domain

The more stable a layer, the longer are the lookups valid (and can be cached longer). As you go downs the hierarchy, you need a faster lookup.

If you make changes to the domain names, it will take typically 2-24 hours for the changes to be visible, for data might be cached.

11.4.4 How the DNS system works?

DNS can be seen as large distributed database that does key-value lookups. The key is machine name and the value is IP address. The table shows the different records. The most common record is A record. MX record mail server responsible to the email service for that machine. CNAME record is for alias. Just as file can have multiple names, a machine can have multiple names as well. You can use nslookup to look at that.

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

Figure 11.6: the different records

11.4.5 Name resolution in DNS systems

In iterative name resolution, you are given a domain name, you split them into pieces from the root domain all the way to the machine name. And you iteratively go on and look for domain name servers of each of these levels of hierarchy, and you will reach where the one will give you the IP address. You iteratively looking for the name by descending down the hierarches, same thing that happened in the file system cases.

Here is a different way to do this called recursive name resolution. You just gave the name to the domain server it figures out what the next name server is and it will forward that request down the tree. Request going down to the domain naming server at each level, until you find the machine. And the response is return up the tree. And then the response comes back to the request server.

We can see that there are pros and cons:

Doing in recursive way, the client first makes one request and get only one response. It reduces the round time request. In iterative way the round trip time for each of the request is going to be large. But recursive will cause heavy load on the name server, especially when we think about there are millions of naming request.

So, if you have long distance between the client and the name severs, the recursive way should be better from performance standpoint. From infrastructure standpoint, in the recursive case node is much heavier loaded in that hierarchy. So the machine should be very powerful to handle very large amount of request. Which is not the case in iterative.

The reason that you may not have the same amount of iterative is that you can actually cache the previous request that have been resolved at your local machine on the local name server. So by caching in the client, you can reduce the overheads require in name resolution.

Some advanced DNS feature can use

1. Same machine has two A records (star.cs.vu.nl) with two IP address. Such thing can be used to clustering webservers. Browser can connect to either of the two. Any IP address can give you the service you want. It can be used to achieve load balance.

2. DNS can make use intelligent of who is making the DNS request and returning different responses based on where is the nearest sever who service that request. So the response time will be lower.

11.4.6 Another naming service: X.500

DNS provides pure key-value lookup. You specify machine names and get IP address. You are not going to do any advanced lookup like general queries. Like if you ask show me all entities that match this attribute, then you want a lists of these entities shown. But DNS cannot do that kind of lookup.

X.500 allows you do more generalized lookup. You can generally ask for any matching attribute. And then it is going to show you all the entities with that. X.500 is the standard that was developed to perform a more advanced directory services.

What is now used more widely is lightweight directory access protocol. It is a more specialized form of X.500 runs on TCP/IP with some restriction but essentially with the same concept. It is widely used in most organization. Like Umass people finder.