

## Project 4: Streaming Audio and Microphone

---

CSC 790

WAKE FOREST  
UNIVERSITY

Department of Computer Science

Fall 2009

### Project 4 Overview

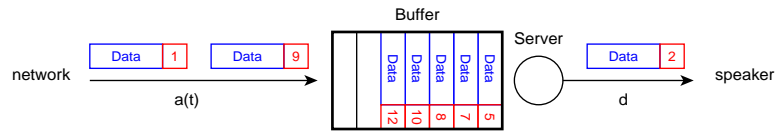
---

- RTP-*lite* Network Karaoke with Flow Control
  - Client/server programs
  - Server streams requested audio
  - Client implements jitter buffer and plays stream mixed with mic
  - Make certain server does not overflow client (flow control)
- Will use RTP-*lite* for sending data

## Jitter Buffers and UDP

- Jitter buffer can be used to provide smooth play-out
  - Initially buffer packets (1 seconds worth))
  - Buffer allows time to reorder packets and determine play-out

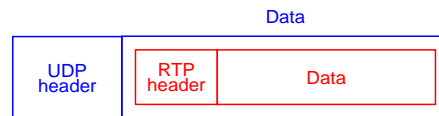
*How are packets ordered in the queue?*



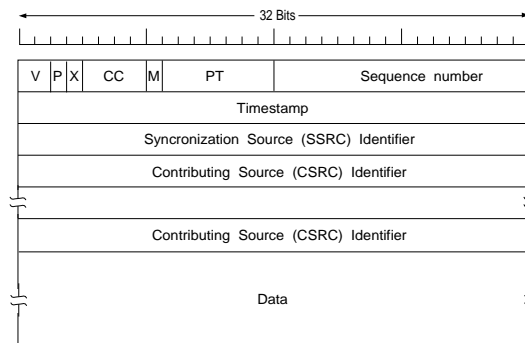
- Multimedia is sent using UDP
  - UDP does not include timers or sequence numbers
  - Need another protocol...

## Real-time Transport Protocol

- Real-time Transport Protocol (RTP) built on top of UDP

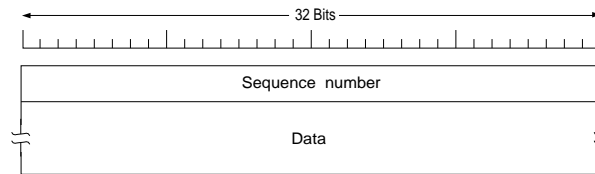


- RTP packet contains: timestamp, sequence number, synchronization, media data, ... however, **no packet length**



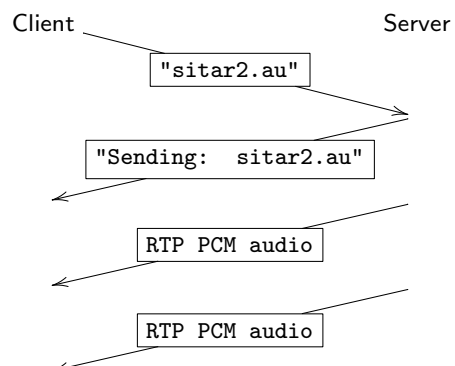
## RTP-lite

- RTP-*lite* also built on top of UDP
- Header contains a sequence number
  - Unsigned long integer (4 bytes)
- Data is variable length (there is a maximum)
  - For this project, data → PCM samples
  - Data should consist of an even number of bytes...



## Network Karaoke

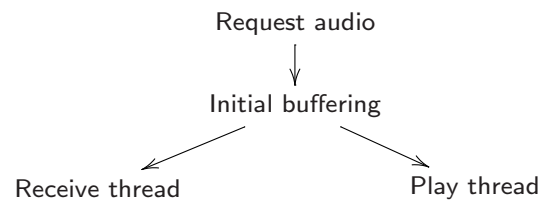
- Client requests one audio file
- Server sends stream (file mixed with microphone) using RTP-*lite*
- Client plays stream, **buffering is required**
  - Must detect lost packets and reorder packets
  - Mix client microphone





## Client Design

- Requests audio stream
- Initially buffers arriving packets (*1 second worth*)
- Receives, buffers, plays packets
  - Must receive and play simultaneously
  - Program must be multi-threaded



*Why can't single threaded program receive then play in series?*

- Receive thread
  - Receives packets from socket
  - Re-orders packets based on sequence numbers
- Play thread
  - Removes packets from queue
  - Send PCM data to speaker
  - **Only send data to speaker when it is needed**, once data is sent to speaker cannot re-order...
- Both threads will access the packet queue, must use semaphores

## Initial Client

```
#include<pthread>    // Unix threads
// other include files ....

// anything needed by threads should be global
UDPsocket          sock;          // socket
PacketQueue       packetQueue;   // packet queue
pthread_mutex_t   queueMutex;    // semaphore for queue
pthread_t         recvThrd;      // receive thread
pthread_t         playThrd;      // play thread

void* recvPacket(void* x);       // function receive thread
void* playPacket(void* x);       // function play thread
```

```
//===main =====
int main(int argc, char* argv[])
{
    // code for requesting audio

    // code for initial buffer

    // initial audio buffered, initialize semaphore and start threads
    pthread_mutex_init(&bufferMutex, 0);
    // create receive thread
    pthread_create(&recvThrd, NULL, recvPacket, NULL);
    // create play thread
    pthread_create(&playThrd, NULL, playPacket, NULL);
    // wait until receive and play threads done
    pthread_join(recvThrd, NULL);
    pthread_join(playThrd, NULL);
    return 0;
}
```

```
//==recvPacket function=====
void* recvPacket(void* x)
{
    pthread_mutex_lock(&queueMutex);
    // insert packet in queue
    pthread_mutex_unlock(&queueMutex);
}

//==playPacket function=====
void* playPacket(void* x)
{
    pthread_mutex_lock(&queueMutex);
    // remove packet from queue
    pthread_mutex_unlock(&queueMutex);
}
```

## How Do You Represent an RTP-lite Packet?

- Packet has few operations, use struct

- Store sequence number and data

```
const int maxSize = 2000; // maximum data size
//==RTP-lite Packet=====
struct RTPLpacket
{
    unsigned long seqNum; // packet sequence number
    char data[maxSize]; // packet data
};
```

- Consider overloading assignment and logic operators

## How Do You Create a RTP-lite Packet?

```
RTPLpacket* createPacket(char* data, int dataLength, int &pktLength)
{
    static unsigned long currentSeqNum = 0; // sequence number
    // check if data > possible packet payload
    if(dataLength > maxSize)
        dataLength = maxSize;

    // create new RTPLpacket
    RTPLpacket* pktPtr = new RTPLpacket;

    // assign sequence number and copy data
    pktPtr->seqNum = (currentSeqNum++);
    memcpy(pktPtr->data, data, dataLength);

    // determine packet length
    pktLength = 4 + dataLength;

    return pktPtr;
}
```

## How Do I Send Data to Speaker, Just in Time?

```
#include<unistd.h> // usleep()

int nSamples = 0; // number of PCM samples read
AuType auFile("file.au");
auFile.readSamples();
while(!auFile.endOfFile())
{
    speaker.write(auFile.pcmSamples(), pcm.numberofSamples());
    nSamples += pcm.numberofSamples();

    // if a second worth of samples sent to speaker, pause a second
    if(nSamples > 8000)
    {
        usleep(int(1*1e6)); // argument in microseconds
        nSamples = 0;
    }
    auFile.readSamples();
}
```

## Flow Control

---

- Providing flow control
  - Easy for server to send too many packets to client
  - Have server send pause if the client side becomes congested
  - Feedback from the client regarding what has been played
  - Therefore the server is also multithreaded
  - Need another socket for feedback (`musicPort + 10`)
- *How long and how often should the server pause?*
  - Maximum buffer size **must** no more than 2 seconds of audio
  - Client must indicate the amount of available space

## Simulating Network Conditions and the **Evil Server**

---

- Client program must correctly detect lost and out-of-order packets
- May not have many errors when transmitting
- Use *evil server* (`evlSrv`)
  - Randomly send packets out-of-order and delays transmission
  - Located at the course web site (*will be*)
  - Program requires audio file as command line argument

## Project 4 Advice

---

- **Start early**
- No late work accepted