REAL-TIME WEB DEVELOPMENT

PART 2

Jim Paterson
James.Paterson@gcu.ac.uk

Peer-to-peer communication

- Chat example in previous lecture may look “peer-to-peer” as it allowed users to chat by sending messages that are immediately seen by other
- However, it is not actually P2P as all messages are sent to or received from an intermediary server
- **WebRTC** is a framework that provides native browser support for true P2P communication where data flows directly from browser to browser
- Supports any data but focus is on video, audio
- As with WebSockets, its use requires browsers that are modern enough to support it

WebRTC

- A WebRTC-enabled application needs to:
  - obtain an audio, video or other data stream
  - gather network information (e.g., IP addresses and ports), and exchange this with other WebRTC clients
  - exchange information about media, such as resolution and codecs
  - stream the audio, video or data
- While data streams from client to client directly, need **signaling** communication, via a server, to initiate or close sessions and report errors – may use WebSockets
- Rather like the process used in traditional telephony to connect calls

WebRTC APIs

- WebRTC applications are based on JavaScript APIs:
  - **RTCPeerConnection**
  - represents a WebRTC connection between the local computer and a remote peer
  - used to handle efficient streaming of data between the two peers.
  - Both peers need to set up their own RTCPeerConnection instances to represent their end of the peer-to-peer connection
  - **RTCDataChannel** - represents a network channel which can be used for bidirectional peer-to-peer transfers of arbitrary data. Every data channel is associated with an RTCPeerConnection
  - **MediaStream** - represents a stream of media content. A stream consists of several tracks such as video or audio tracks.

WebRTC examples

- Here we will simply look at some code fragments that illustrate how these APIs can be used
- Full implementations of WebRTC in “real-world” network environments can be quite complex
- For useful full examples, see:
  - https://codelabs.developers.google.com/codelabs/webrtc-web/#0
  - https://www.codeproject.com/Articles/1073738/Building-a-Video-Chat-Web-App-with-WebRTC

RTCPeerConnection

- The initiator of the call (the caller), needs to create an offer and using a signaling service (e.g., a NodeJS server application using WebSockets) send it to the callee:

```
var peerConn= new RTCPeerConnection();

peerConn.createOffer(function(offer) {
  peerConn.setLocalDescription(new RTCSessionDescription(offer),
    function() {
      // send the offer to a server to be forwarded to other peer
  }, error);
}, error);
```

- The callee, which receives the offer and needs to "answer" the call has to create an answer and send it to the caller, using peerConn.createAnswer

RTCDataChannel

- Objects of this type can be created using RTCPeerConnection.createDataChannel()

```
var peerConn= new RTCPeerConnection();
dc = peerConn.createDataChannel("my channel");
dc.onmessage = function (event) {
  console.log("received: " + event.data);
};
dc.onopen = function () {
  console.log("datachannel open");
};
dc.onclose = function () {
  console.log("datachannel close");
};
```


MediaStream

- Create with navigator.getUserMedia method
- Can add stream to an RTCPeerConnection

```
navigator.getUserMedia({ audio: true, video: true},
  function(stream) {
    var video = document.getElementById('localVideo');
    video.src = window.URL.createObjectURL(stream);
    peerConn.addStream(stream);
  },
  function(err) {
    console.log("The following error occurred: " + err.name);
  }
);
```

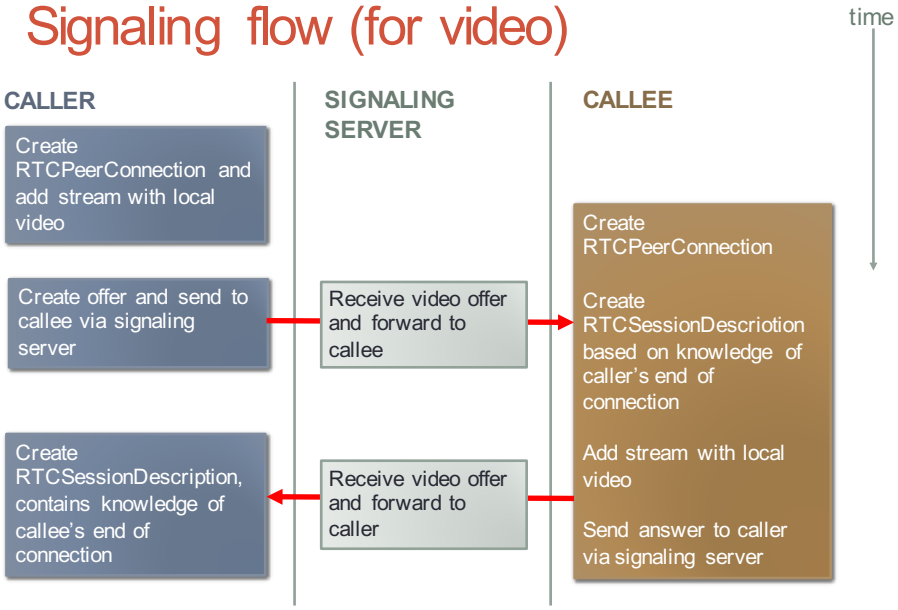
Signaling

- Communication between browsers with WebRTC is (usually) peer-to-peer
- However, in order to set up peer connection we need a **signaling channel**
- Once call is set up by signaling, direct communication can start
- Analogous to the operators in old telephone exchanges who physically connected calls

Signaling server

- Need a **signaling server** to allow exchange of session descriptions and network reachability
- Communication between browsers and signaling server often done with WebSockets
- Offer contains session information, using **Session Description Protocol** (SDP)
- Using signaling server and SDP the two participants know codecs and video parameters to be used for this call

Signaling flow (for video)



Dealing with real-world networks

- Direct communication between peers is straightforward if they are on the same network
- Can simply use their IP addresses to indicate endpoints
- This would work inside a corporate or home network, for example
- However, communication with remote peers, on different networks, in the real world is more complicated
- Corporate networks use firewalls and gateways at network boundary
- Home networks do the same, where the home router is the gateway between devices in home and the ISP network

IP addresses and NAT

- A **public IP Address** is an IP address that is globally unique across the Internet
- A **private IP Address** is an IP address that is not globally unique and may exist simultaneously on many different devices
- A private IP address is never directly connected to the Internet
- **Network Address Translation** (**NAT**) gives private IP addresses access to the Internet
- NAT allows a single devices, such as a router, to act as an agent between the Internet (populated with public IP addresses) and a private network (populated with private IP addresses)

13

ICE

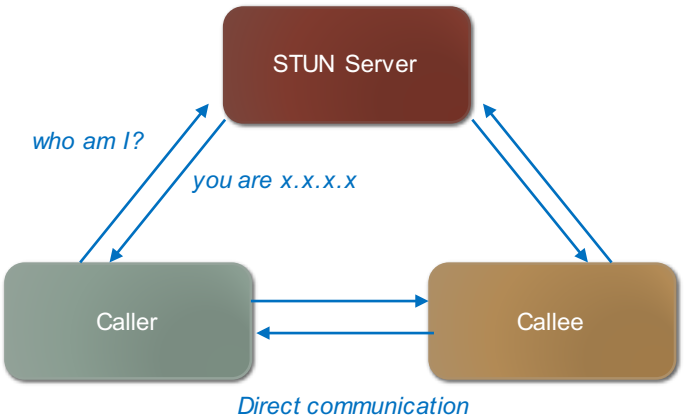
- WebRTC clients typically need communicate with each other when neither has an IP address that the other can send directly to
- **Internet Connectivity Establishment (ICE)** is a framework that allows WebRTC to overcome the complexities of real-world networking
- Finds the best path to connect peers
- May be able to do that with a direct connection between the clients, but it also works for clients where a direct connection is not possible (i.e. behind NATs)
- Uses two approaches depending on network configuration – STUN and TURN

14

STUN

- Some NATs map a private IP address to a consistent public IP address to allow communication from outside the network
- ICE can use a **Session Traversal Utilities for NAT (STUN)** server
- Allows clients to discover their public IP address and the type of NAT they are behind
- This information is used to establish the media connection
- In most cases, a STUN server is only used during the connection setup and once that session has been established, media will flow directly between clients.

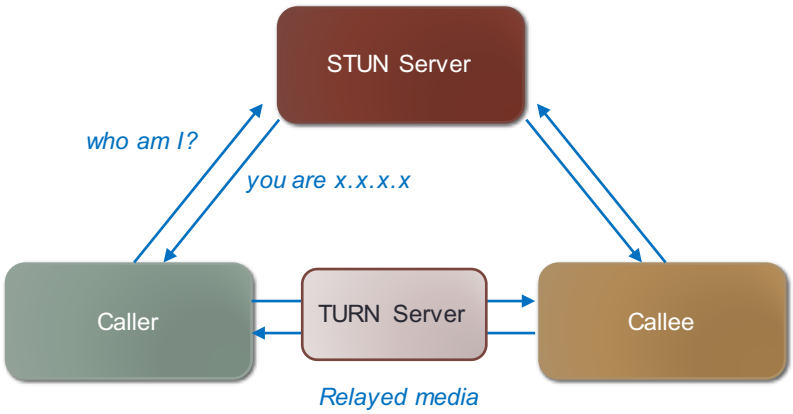
STUN



TURN

- Sometimes STUN is not sufficient to allow communication, for example where the NAT does IP address and port translation
- This the case for a variation of NAT called **symmetric NAT**
- **Traversal Using Relay NAT (TURN)** is an extension to STUN that can be used when basic STUN doesn't work
- Unlike STUN, a TURN server remains in the media path after the connection has been established
- A turn server is often referred to as a **relay** server

TURN



Using STUN/TURN

- Need to tell your WebRTC application where to find the STUN and TURN servers
- Do this when configuring the RTCPeerConnection
- Preferable to use STUN but WebRTC applications must be prepared to support TURN also
- There are some public servers which are useful for prototyping and non-critical applications, e.g. Google has public STUN servers
- Public TURN servers are less common and support authentication
- Note that ICE is mechanism that works **alongside and in addition to the signalling channel** for WebRTC

Summary

- *Peer-to-peer communication and WebRTC*
- *WebRTC APIs*
  - *RTCPeerConnection*
  - *RTCDataChannel*
  - *MediaStream*
- *Signaling and signaling servers*
- *ICE*
  - *STUN*
  - *TURN*