# WebRTC

How we've been doing lectures
this last few weeks...

Various bits of this follow:

https://github.com/webrtc-security/webrtc-security.github.io/blob/master/index.md

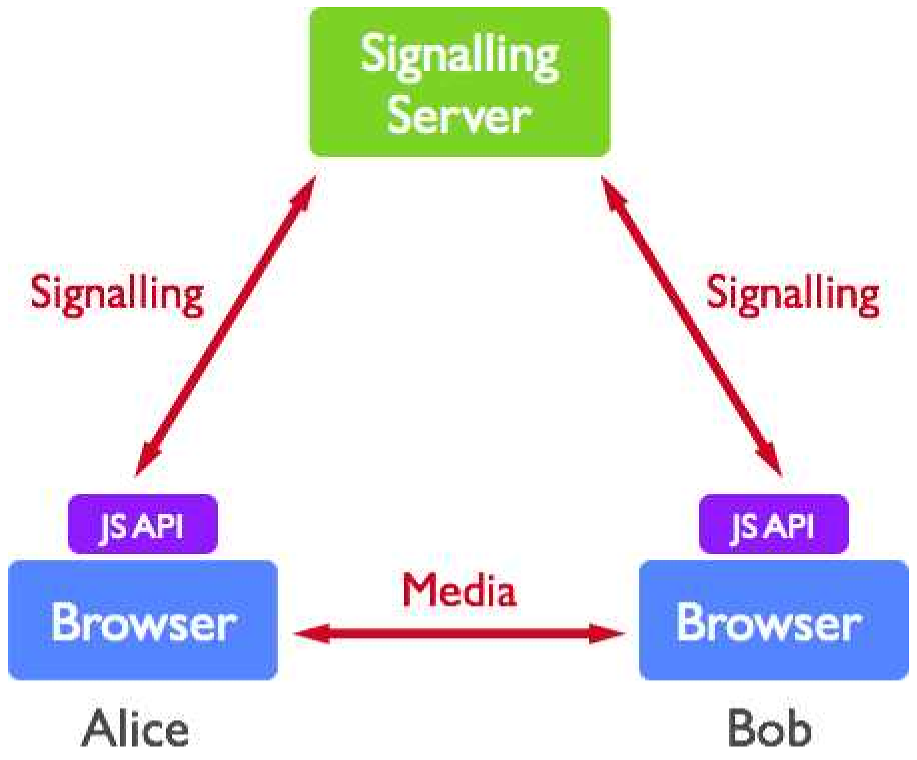
# WebRTC Functions

- WebRTC is a way to make encrypted voice/video calls **directly** between two web browsers
  - Ideal flow is direct ciphertext packets from one browser to the other and that works just fine e.g. with IPv6 at home, that is; in a **peer-to-peer** mode
- A web site mediates call setup using Javascript and standard interfaces implemented in web browsers, and calls use standard protocols for data, voice and video
- Uses: sales and support for web sites, conference calling for enterprises, calling between their subscribers

# WebRTC Standardisation

- Co-operation between IETF (protocols) and W3C (browser APIs), largely driven by browser-makers but with lots of input from more traditional telcos
  - https://datatracker.ietf.org/wg/rtcweb/
  - https://www.w3.org/2018/07/webrtc-charter.html
- Mostly smooth but took way longer than expected due to inherent complexity and a major bun-fight (>2 years) about which audio/video codecs to make mandatory-to-implement
  - Such codecs have historically been patented – the telco industry was fine with that but with WebRTC open-source browsers replace handsets so patents and odd IPR licenses become much more of an issue
- Standardisation-nerd lesson: don't create a huge web of interdependencies in specifications – approx 100 almost-finished RFCs have been queued up waiting on one another for up to 5 years!
  - https://www.rfc-editor.org/cluster_info.php?cid=C238

# WebRTC Basic Flow



# WebRTC – the bad side

- Web site that sets up call can eavesdrop, maybe beyond life of call
  - Can be detected in many cases, so less likely to be done commonly, but there are many bad actors in the world
- “Consent” model is as broken as ever, but WebRTC extends SOP security model and “consent” model (for camera/mic) so is therefore making a bad situation worse
- Peer-to-peer connections (a good thing) require exposing information about the peers’ IP addresses to work – sometimes those addresses can be sensitive
  - OTOH WebRTC probably exposes many new correlators/identifiers despite the designers’ best intentions
  - And they have tried to eliminate new privacy leaks where possible

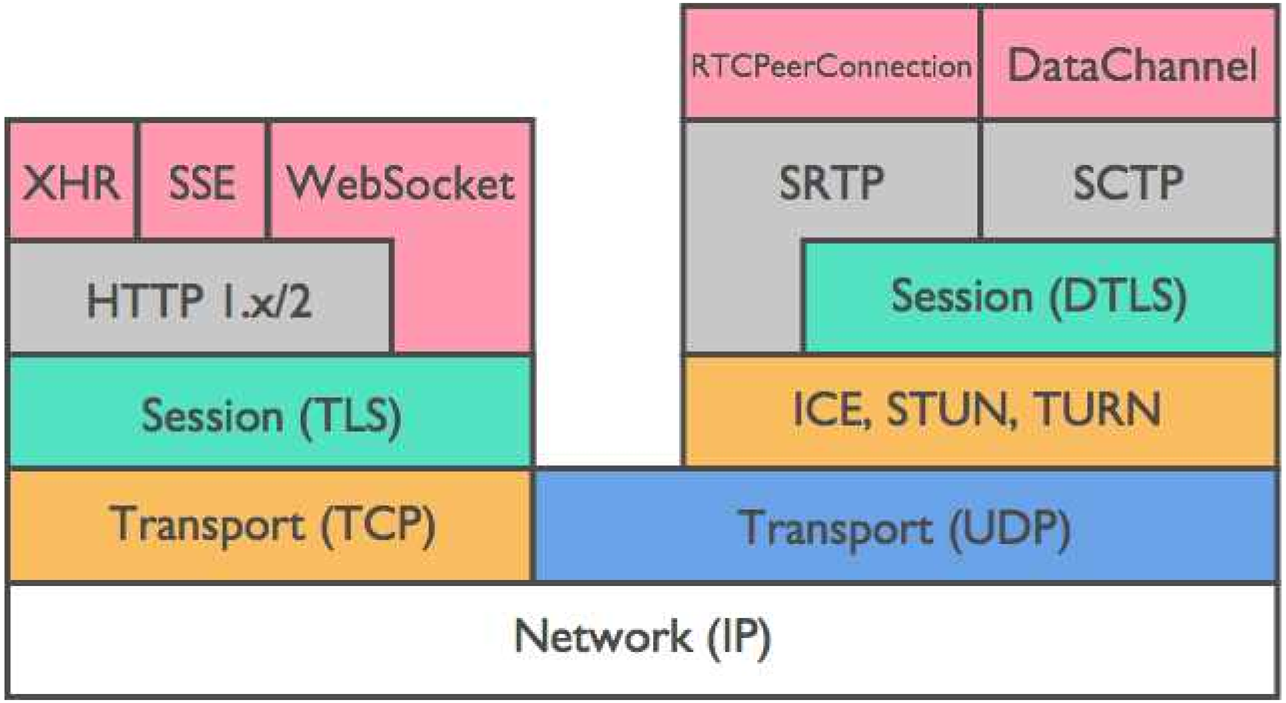
## WebRTC – the good side!

- You can be your own phone company, https://jell.ie/webrtc/  is mine
  - That's all built from open-source, I didn't write any WebRTC code at all, all I did was download, build and configure stuff, and I didn't have to ask anyone for permission (figuring out how is time consuming but doable)
- https://meet.jit.si/somerandomname is a slightly more scaleable WebRTC deployment and various others exists at that scale
- Highly scalable tools like Cisco's Webex use WebRTC without needing special client s/w; I've been on "calls" with >200 people that replaced a cancelled f2f conference session
- Blackboard's "collaborate ultra" feature is also a WebRTC instance
- In COVID19 lockdowns, WebRTC has allowed many, many people to continue to work from home without having to install weird conference calling clients (not all video calling tools use WebRTC)

## WebRTC Protocol Stack

- The WebRTC protocol stack is incredibly baroque and complex
  - All the usual HTTPS stuff, plus...
  - Weird mixture of DTLS, SCTP, UDP, SRTP
  - With added ICE, STUN and TURN for NAT fun
  - And SDP for offer/answer fun
- Reasons for WebRTC complexity:
  - Browsers don’t normally talk to one another
  - TCP isn’t great for real-time media
  - Negotiating media parameters (codecs, bit rates, screen details) is complex
  - Complicated corner-case use-cases: legacy interop, conference calls

## WebRTC Stack



## WebRTC API Terms

- XHR – XMLHttpRequest is a (now outdated by fetch()?) way to write asynchronous JS code
- SSE – Server Sent Events API (uncommon?)
- WebSockets – provide a way to send arbitrary data from a browser to a peer
- RTCPeerConnection – is a new WebRTC API that allows a browser tx/rx real-time data with a peer browser in a WebRTC session
- DataChannel is another WebRTC API, but for non-realtime data, e.g. the chat session in a call

## SDP for media

- Session Description Protocol (SDP, RFC 4566) was designed before WebRTC for VoIP
- Offer/answer model where one peer says e.g. “I can do AV8, screen: 1024x768” etc. and other peer answers – kind of like TLS ciphersuite negotiation but with many more degrees of freedom
  - Developers generally hate SDP as it’s exposed to the JS coder

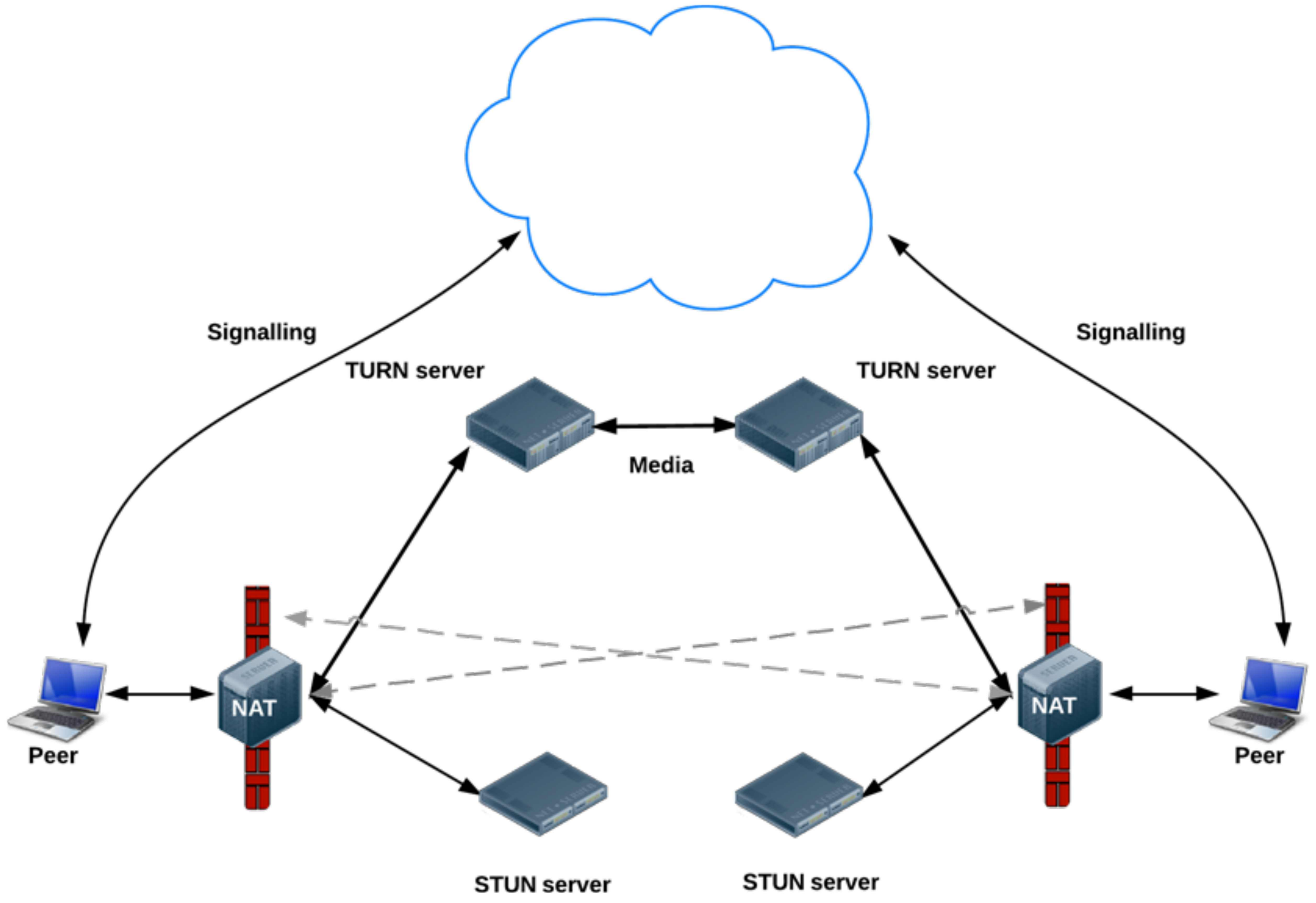
# WebRTC Protocol Terms

- SRTP – is the Secure Real Time Protocol, RFC 3711 defines a way to secure real-time media traffic
- DTLS – is the Datagram Transport Layer Security protocol, RFC 6347
- DTLS-SRTP, RFC 5764 – defines how to use DTLS session keys for SRTP
- SCTP – is the Stream Control Transmission Protocol, RFC 4960 – basically provides a mix of TCP-like and UDP-like features

# WebRTC and NATs

- Many browser instances will be behind a NAT
- ICE – Interactive Connectivity Establishment (RFC 8445) is an algorithm for picking/agreeing whether to communicate directly or not, and if not, how – may use STUN and/or TURN
- STUN – Session Traversal Utilities for NAT (RFC 5389) – a STUN server on the public Internet enables a browser to figure out it’s public IP address
- TURN – Traversal Using Relays around NAT (RFC 5766) provides a way to relay packets if there’s no other way to get them inside the NAT
- Mostly: TURN servers are also STUN servers, but STUN servers are cheap (no media) whereas TURN servers can be expensive (media traffic goes via TURN server), so many STUN servers might not (willing to) offer TURN service

# WebRTC and NAT



From https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

## WebRTC Security Model

- Browser SOP extended – user is prompted to allow access for **site** to **microphone/camera** and that permission is **sticky** if the site uses HTTPS
  - When WebRTC started, HTTPS was not as ubiquitous as today
- Use of microphone/camera likely indicated via icon on browser bar, and that's also how a user can **revoke permission** (if they remember – would they?)
- People on phones have less control in reality
- Overall effect: web sites get permission is my bet – not sure if anyone has tried to measure