Aalto University
School of Science
and Technology

Performance analysis of topologies for Web-based Real-Time Communication

Albert Abello Lozano

Supervisor: Jörg Ott
Instructor: Varun Singh
Communication and Networking Department
Aalto University, School of Science and Technology
albert.abello.lozano@aalto.fi

Espoo , May 22, 2013

Outline

- Introduction
- Background
- WebRTC
- Topologies
- Evaluation environment
- Tests and results
- Conclusions

Introduction

- Need of connectivity between people
- Real-time communication is in our daily life
- Complex interactive web applications are everywhere
- Developers need a way to handle real-time cross-interoperable apps in the web

Background

- Open sourced by Google
- First introduced in May 2011
- Designed in conjunction by the IETF and W3C
- Still in ongoing discussion
- Works using high level JavaScript APIs
- First version to be delivered by Q4 2013
- Uses existing technologies derived from SIP

Support

Google™

Firefox

CISCO™

™

ERICSSON

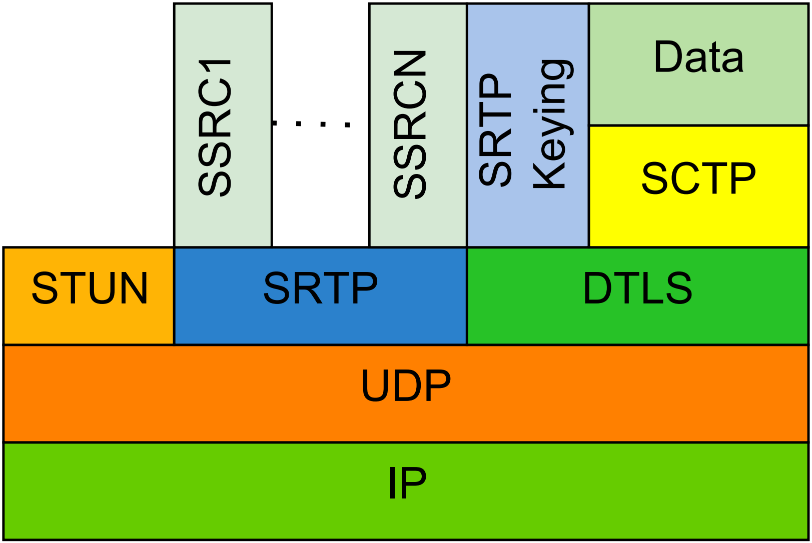
WebRTC

- Provides real-time peer-to-peer media and data transport using browsers
- Plugin-free, mechanisms are integrated on the browsers
- Interoperable between vendors
- Uses JavaScript APIs to enable the features
- Federated domain video calls
- *Google Chrome* and *Mozilla Firefox* implement all the APIs
- *Opera* only includes *GetUserMedia*

WebRTC network internals

- RTP and RTCP multiplexed over the same port
- SDP for signaling content
- SCTP and DTLS for secure data
- SRTP for media
- STUN, TURN and ICE for NAT reversal
- All traffic over UDP

WebRTC protocol stack

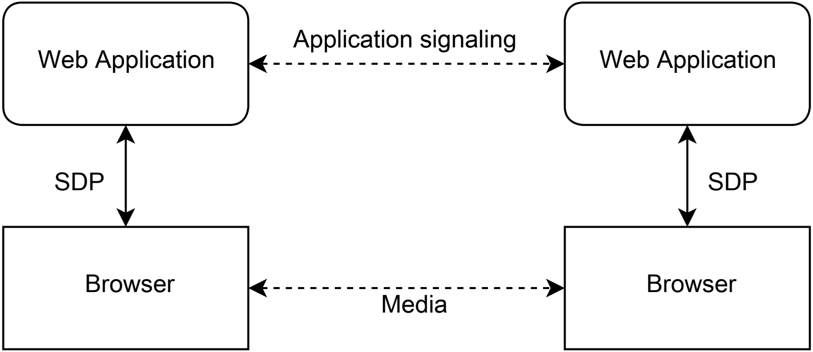


WebRTC implementation

- VP8 as de facto video codec
- G711 and Opus as audio codec
- H.264 video codec in some browser vendors
- Uses JSEP for signaling

Media codecs are still on an ongoing discussion

JSEP signaling model



WebRTC APIs

- Combines two high-level JavaScript APIs:
  - *GetUserMedia()*
  - *PeerConnection()*
- Uses different specific objects:
  - *MediaSream*
  - *MediaStreamTrack*
  - *DataChannel*
- Control and monitoring:
  - *getStats()* method is used for monitoring
  - JSON objects change the constraints for video and networking
- *Google Chrome* and *Mozilla Firefox* implement all the APIs
- *Opera* only includes *GetUserMedia*

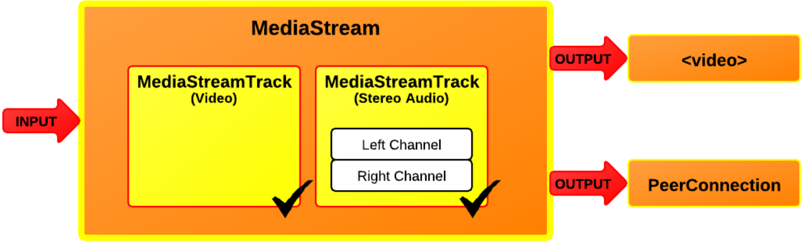
GetUserMedia()

Provides access to media devices and returns a *MediaStream* object.

```
navigator.webkitGetUserMedia(cameraConstraints(),
    gotStream, function() {
   console.log("GetUserMedia failed");
});

function gotStream(stream) {
   //Stream is the MediaStream object returned by the API
       and played in HTML local-video element
   console.log("GetUserMedia succeeded");
   document.getElementById("local-video").src =
       webkitURL.createObjectURL(stream);
}
```

MediaStream **and** *MediaStreamTrack*



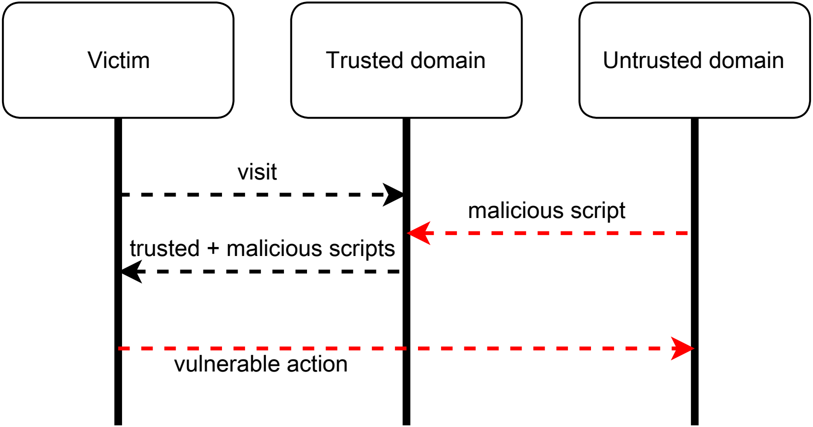
PeerConnection()

Builds a point-to-point connection

```
pc = new webkitRTCPeerConnection(pc_config);
pc.onicecandidate = iceCallback;
pc.addStream(localstream);
function iceCallback(event){
   if (event.candidate) {
      sendMessage(event.candidate);
   }
}
pc.addIceCandidate(new RTCIceCandidate(event.candidate));
pc.onaddstream = gotRemoteStream;
function gotRemoteStream(e){
   document.getElementById("remote-video").src =
       URL.createObjectURL(e.stream);
}
```

Security in WebRTC

JavaScript libraries can provide cross-scripting and trust vulnerabilities



Thank you!