Real-Time communication with WebRTC

Lieven Desmet – iMinds-DistriNet, KU Leuven
Lieven.Desmet@cs.kuleuven.be

SecAppDev Leuven 2015 (27/02/2015, Leuven)

About myself: Lieven Desmet

@lieven_desmet

- Research manager at KU Leuven
  - (Web) Application Security
- Active participation in OWASP
  - Board member of the OWASP Belgium Chapter
  - Co-organizer of the OWASP AppSec EU 2015 Conference
- Program director at SecAppDev

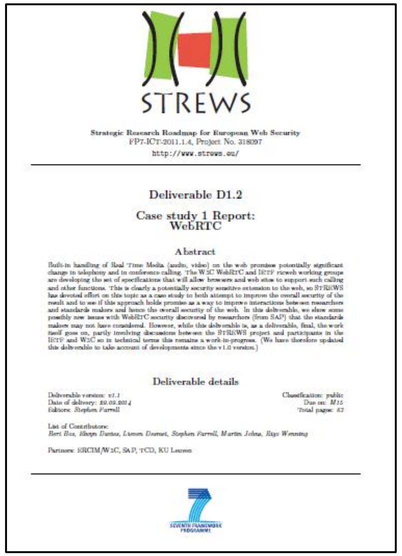
iMinds-DistriNet, KU Leuven

- Headcount:
  - 10 professors
  - 65 researchers
- Research Domains
  - Secure Software
  - Distributed Software
- Academic and industrial collaboration in 30+ national and European projects

https://distrinet.cs.kuleuven.be

Relevant sources

- Large security assessment of relevant specifications
  - Joint work with IETF, W3C and SAP on security of WebRTC
  - https://www.strews.eu/results/91-d12



STREWS

Strategic Research Roadmap for European Web Security
FP7-ICT-2011.1.4, Project No. 318097
http://www.strews.eu/

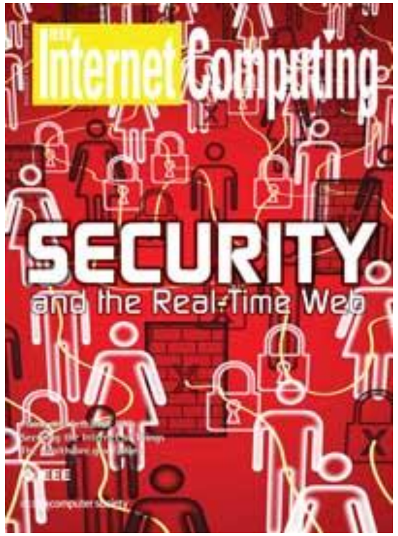
Deliverable D1.2

Case study 1 Report: WebRTC

Abstract

Deliverable details

- Identifying open issues and security challenges for WebRTC
  - Special Issue of IEEE Internet Computing, nov/dec 2014
  - http://www.computer.org/csdl/mags/ic/2014/06/index.html



WebRTC ?

Real-Time communication on the Web

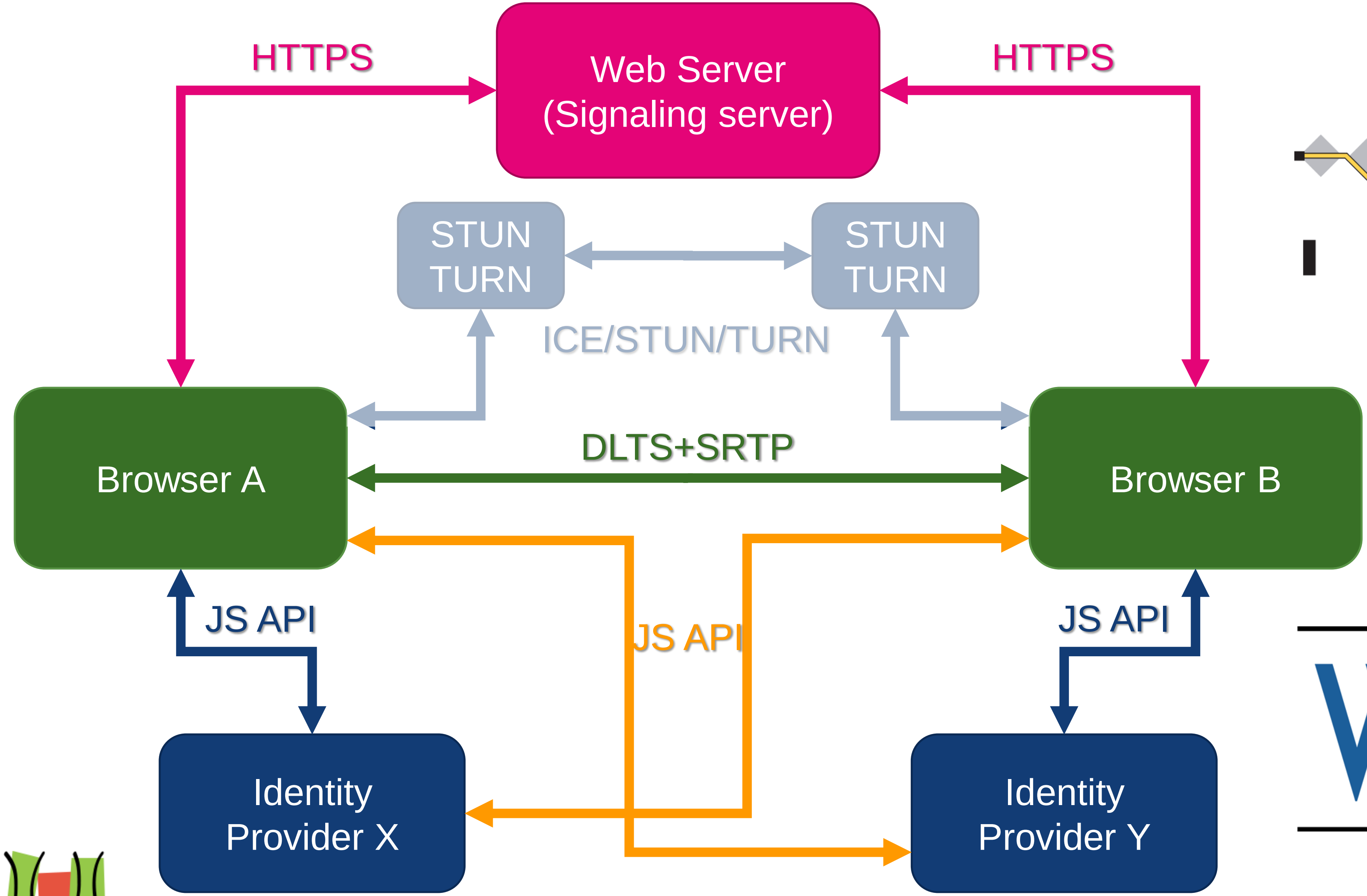

http://youtu.be/MsAWR_rJ5n8

Overview

- Introduction to WebRTC
- WebRTC JavaScript APIs
- WebRTC deployments
- Overview of attack vectors
- Wrap-up

Introduction to WebRTC

WebRTC architecture



Signaling path

- Signaling path between WebRTC end-points
- Signaling server(s)
  - Loads client-side context (JavaScript code)
  - Mediates control messages and meta-data between end-points
- Signaling protocol is undefined in WebRTC
  - Up to the application to deploy one !

Media path

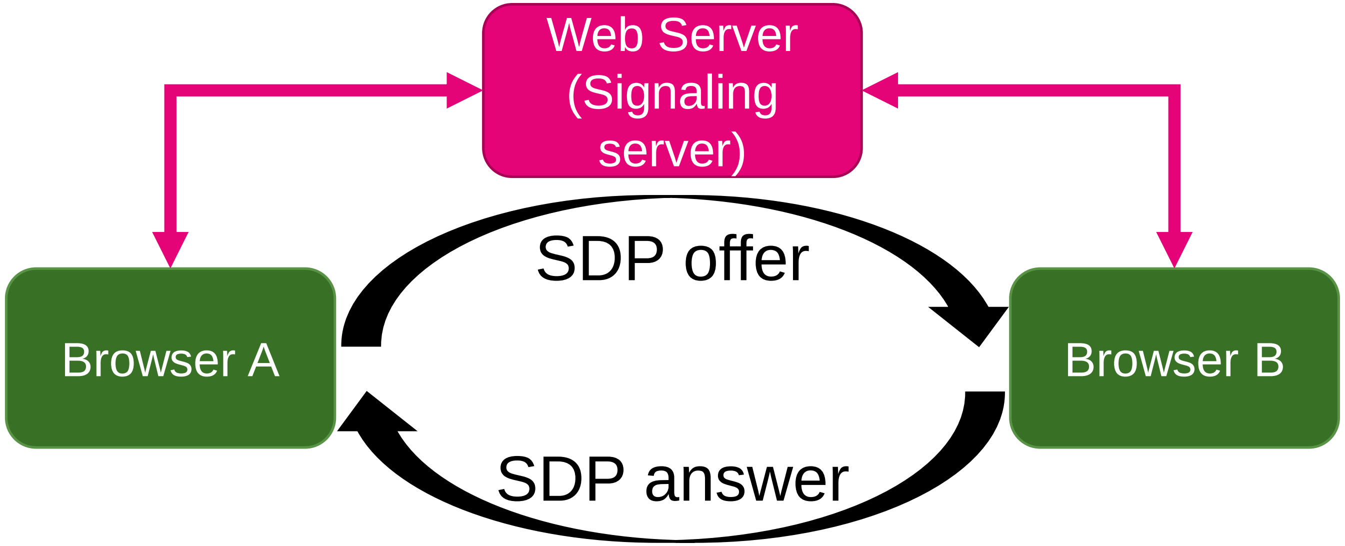
- Secure peer-to-peer connection between browsers
  - Media streams (video/audio)
  - Data channels
- DTLS: Datagram Transport Layer Security
- SRTP: Secure Real-time Transport Protocol
  - Encryption, message authentication and integrity

Setting up the media path

- SDP
- UDP hole punching
- STUN
- TURN
- ICE

SDP: Session description protocol

- Initialization parameters for streaming media
  - Session announcement
  - Session invitation
  - Parameter negotiation (multimedia types, codecs, …)
- SDP offer and SDP answer



SDP example

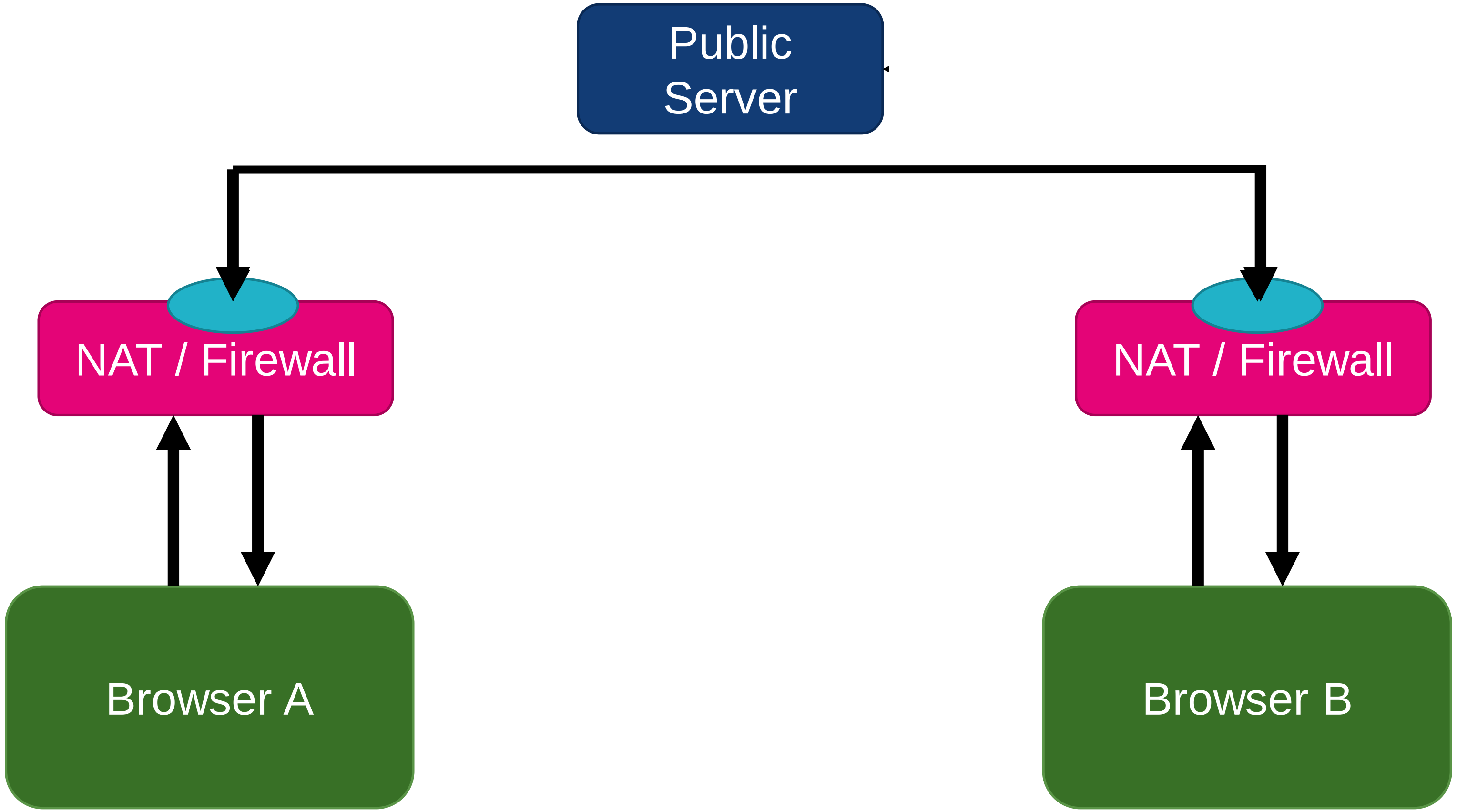
```
v=0
o=- 20518 0 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS ma
a=group:BUNDLE audio
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8
c=IN IP4 24.23.204.141
a=mid:audio
a=msid:ma ta
a=rtcp-mux
a=rtcp:54609 IN IP4
24.23.204.141
a=rtpmap:109 opus/48000/2
a=ptime:60
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
…
```

```
…
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=sendrecv
a=setup:actpass
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:
1f:66:79:a8:07
a=ice-ufrag:074c6550
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068
a=candidate:0 1 UDP  2122194687 192.168.1.4 54609 typ host
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host
a=candidate:1 1 UDP  1685987071 24.23.204.141 64678 typ srflx
raddr 192.168.1.4 rport 54609
a=candidate:1 2 UDP  1685987071 24.23.204.141 64678 typ srflx
raddr 192.168.1.4 rport 54609
a=rtcp-fb:109 nack
a=ssrc:12345 cname:EocUG1f0fcg/yvY7
a=rtcp-rsize
a=ice-options:trickle
```

SDP Offer taken from “SDP for the WebRTC” (IETF Internet Draft)

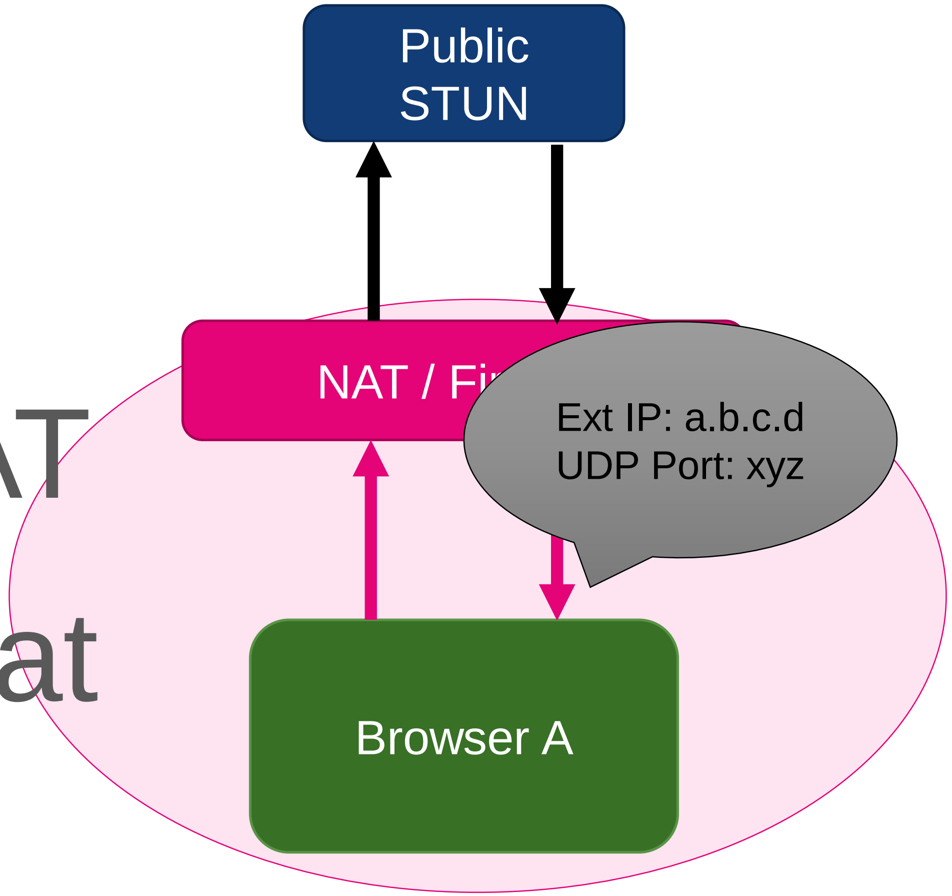
UDP hole punching

- Enables connectivity between peers across NAT(s)



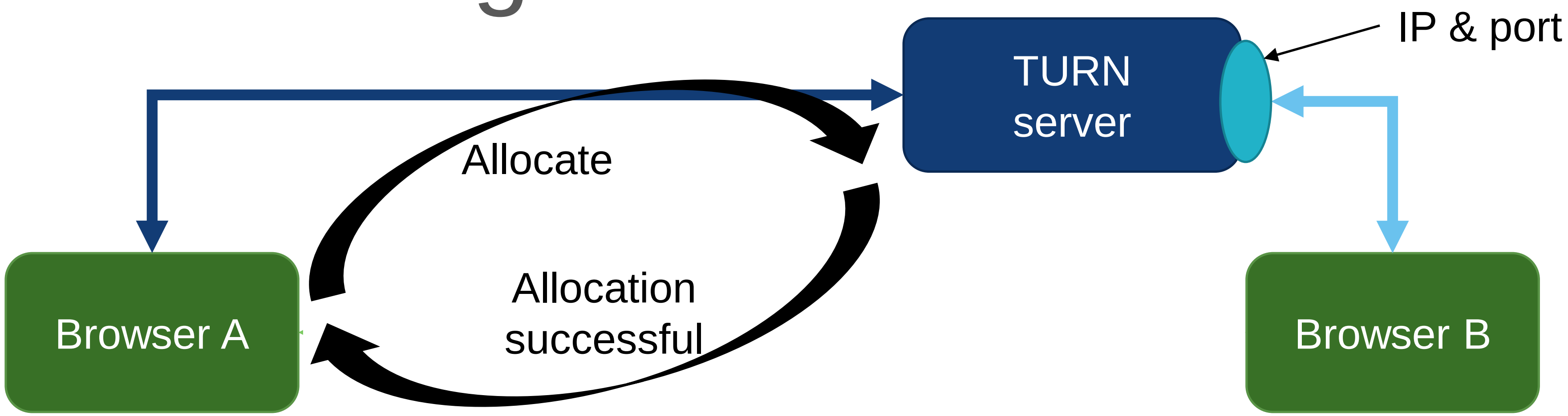
STUN: Session Traversal Utilities for NAT

- Discover your public IP address
- Determine whether your browser sits behind a NAT
- Retrieve the UDP port that NAT has allocated for external communication



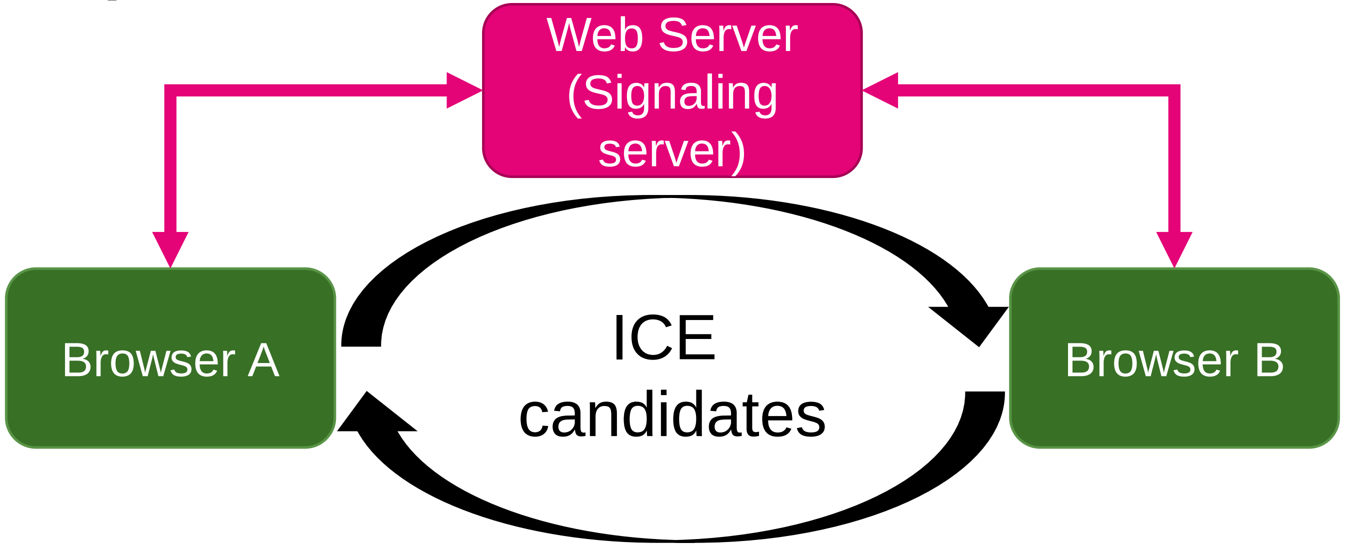
TURN: Traversal Using Relays around NAT

- Used if STUN does not work
- TURN server relays traffic between 2 NAT'ed peers
- IP and port get allocated on STUN for sending or receiving a stream



ICE: Interactive Connectivity Establishment

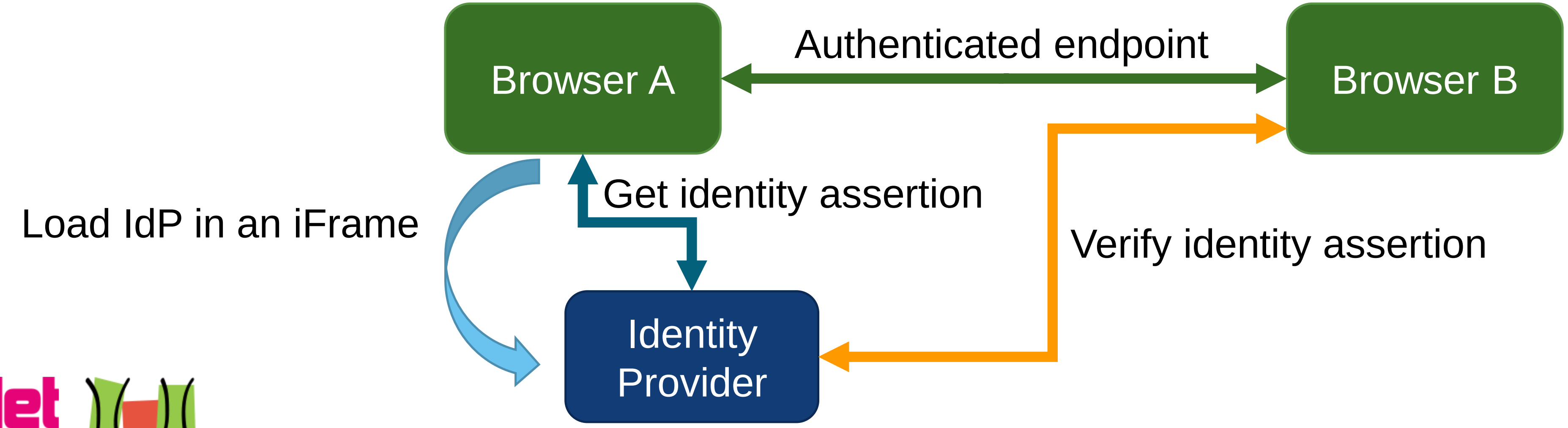
- Gathering info (STUN, TURN, …)
- Offering and answering ICE candidates between peers
- Probe candidates in order of priority
  - Until ICE candidate pair work



DistriNet iMinds KU LEUVEN STREWS

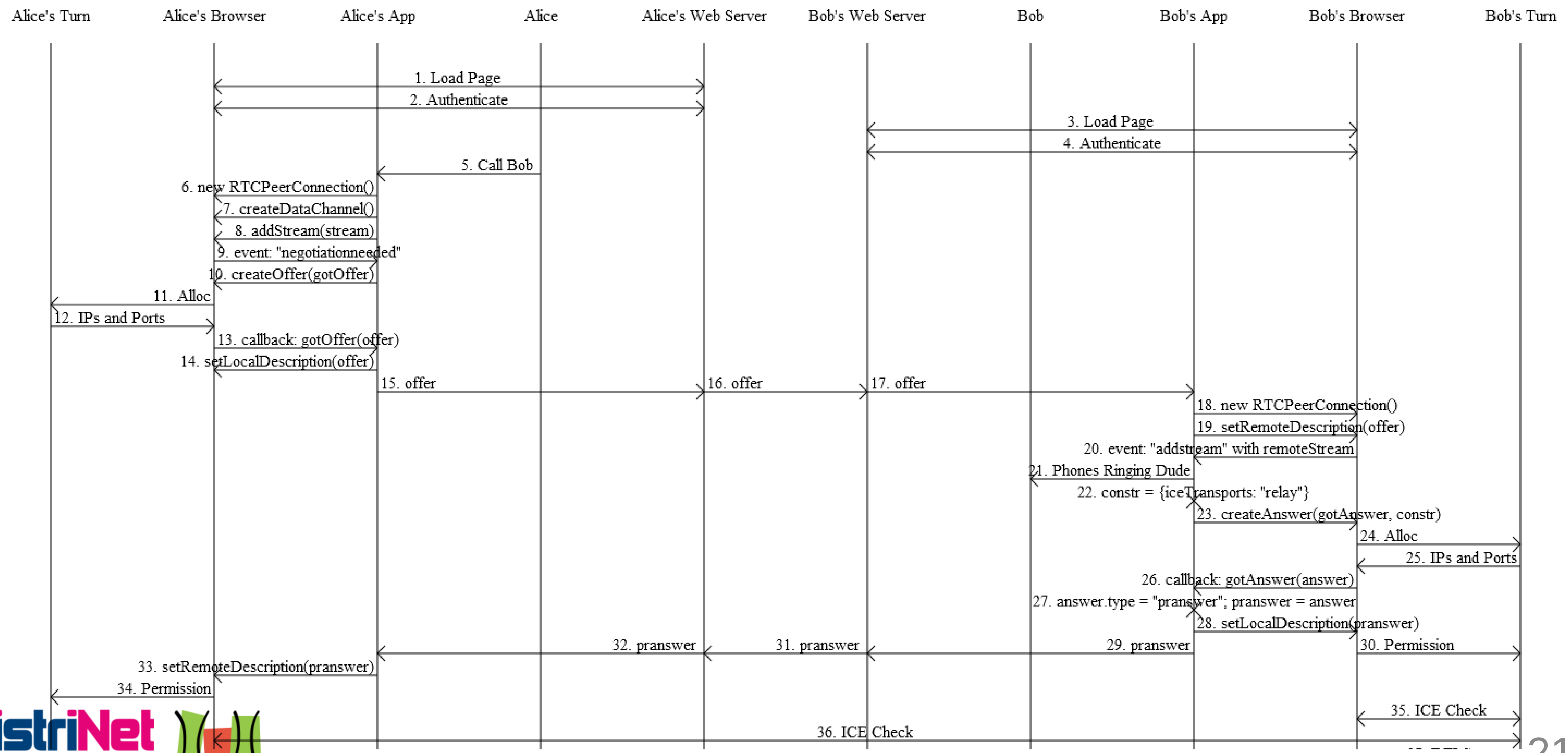
Identity provision

- To authenticate the endpoint, an Identity Provider (IdP) can be used
  - Code of IdP gets loaded in an iframe
  - Interaction between client-side code and IdP via Web Messaging (aka postMessage)



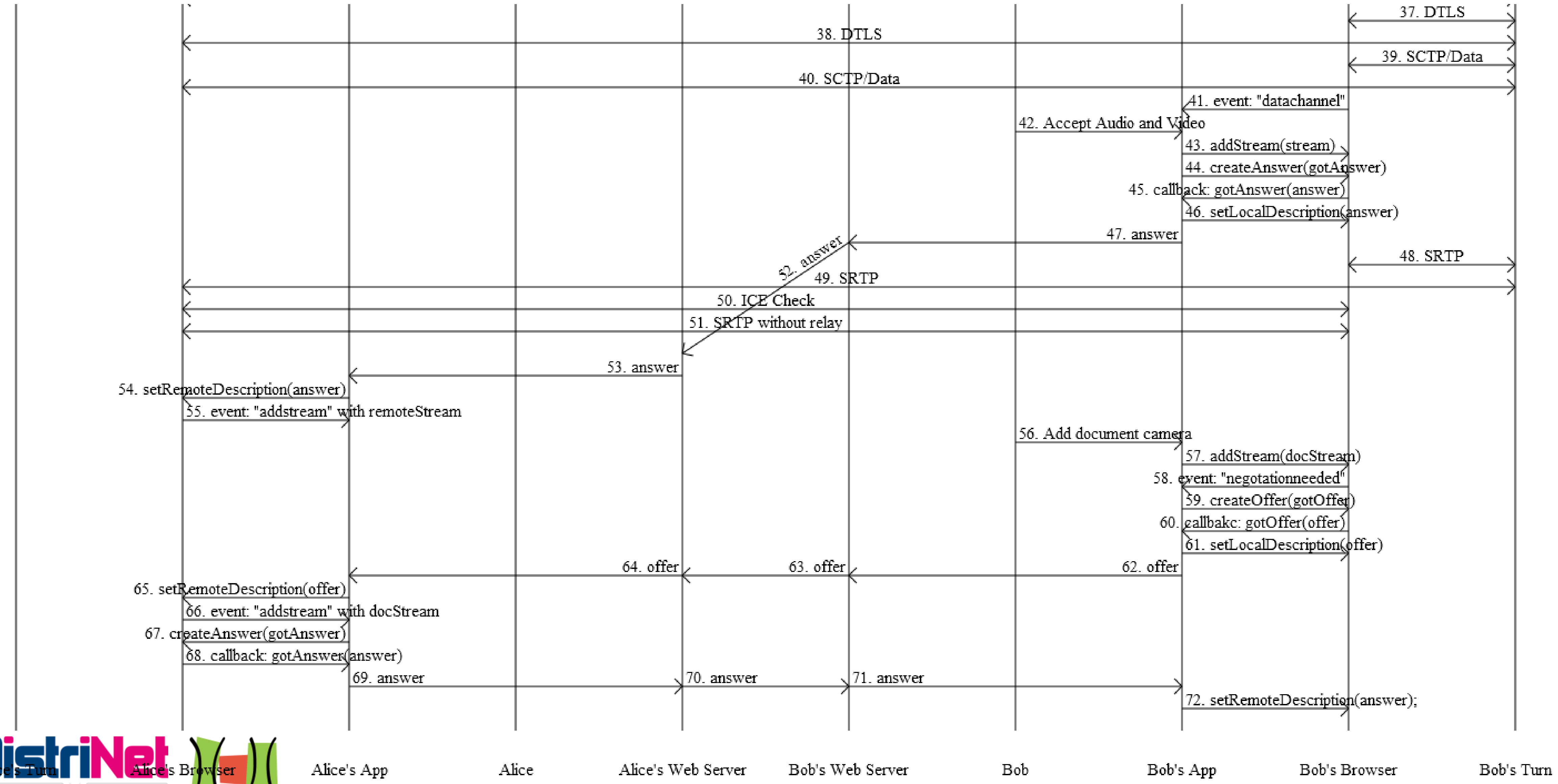
WebRTC JavaScript APIs

To give you an idea of the complexity: The simple case… (1)



Taken from "WebRTC 1.0: Real-time Communication Between Browsers" (W3C Editor's Draft)

To give you an idea of the complexity: The simple case… (2)

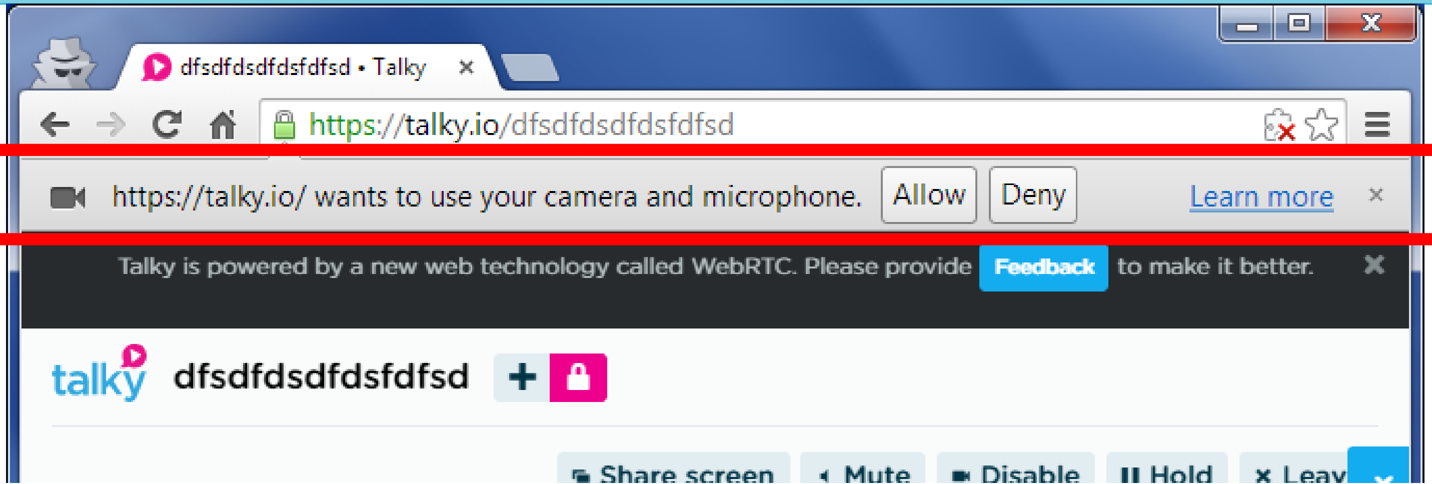


Taken from "WebRTC 1.0: Real-time Communication Between Browsers" (W3C Editor's Draft)

Capturing a video stream

```
// overcome temporary browser differences ☺
navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia ||
navigator.mozGetUserMedia;

// request a UserMedia Stream and use it on the local page and the RTCPeerConnection
navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {
    if(stream){
        video1.src = URL.createObjectURL(stream);
        peerConnection.addStream(stream);
    }
}, logError);
```

Asks the user for permission



DistriNet iMinds KU LEUVEN STREWS

Setting up a RTCPeerConnection

```
// overcome temporary browser differences ☺
RTCPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection ||
window.webkitRTCPeerConnection;

// configuration of STUN, TURN, …
// can also be derived automatically by the browser
var configuration = {
    "iceServers": [{ "url": "stun:stun.example.org" }]
};

// Creating the Connection object and add a handler for incoming streams
peerConnection = new RTCPeerConnection(configuration);

peerConnection.onaddstream = function (evt) {
    video2.src = URL.createObjectURL(evt.stream);
};
```

DistriNet iMinds KU LEUVEN STREWS

Handling SDP offers and answers

```
// create a SDP offer on negotiation
peerConnection.onnegotiationneeded = function () {
    peerConnection.createOffer(function (offer) {
        // set it as the Local SDP description and send the offer to the other peer
        return peerConnection.setLocalDescription(offer, function () {
            signalingChannel.send(JSON.stringify({ "sdp": peerConnection.localDescription }));
}) }) };
```

```
signalingChannel.on('message', function (evt) {
    if(message.sdp){
        var desc = new RTCSessionDescription(message.sdp);
        // if we get an offer, we need to reply with an answer
        peerConnection.setRemoteDescription(desc, function () {
            return peerConnection.createAnswer(function (answer) {
                return peerConnection.setLocalDescription(answer, function () {
                    signalingChannel.send(JSON.stringify({ "sdp": peerConnection.localDescription }));
}) }) }) } });
```

Application-specific signaling!

Handling ICE Candidates

```
// send any ice candidates to the other peer
peerConnection.onicecandidate = function (evt) {
    if (evt.candidate) {
        signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));
    }
};
```

```
// receive and process remote ICE candidates
signalingChannel.on('message', function (evt) {
    if(message.candidate){
        peerConnection.addIceCandidate(new RTCIceCandidate(message.candidate));
    }
});
```

Setting up a data channel

```
// setting up a data channel
var dataChannel = peerConnection.createDataChannel("myLabel", dataChannelOptions);

dataChannel.onerror = function (error) { … };

dataChannel.onmessage = function (error) { … };

dataChannel.onopen = function (error) { … };

dataChannel.onclose = function (error) { … };
```

Identity provision

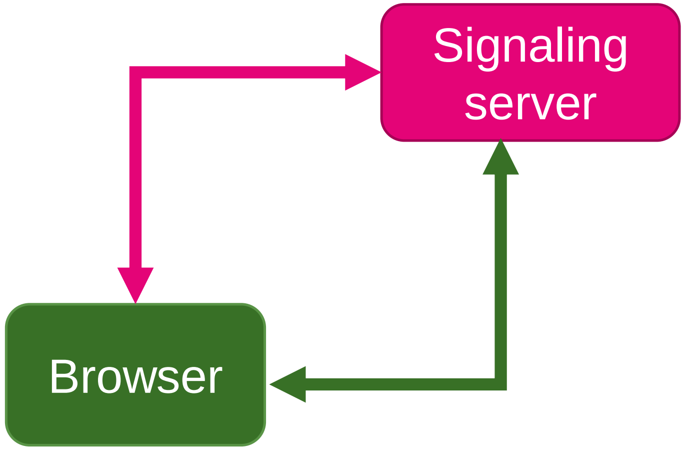
```
// setting up the identity provider
// [ this can also be done by the browser ]
// commented out example: also provide optional protocol and username
// peerConnection.setIdentityProvider("example.com", "default", "alice@example.com");

peerConnection.setIdentityProvider("example.com");

// possible interaction with the IdP proxy
// this is done implicitly by the PeerConnection
peerConnection.getIdentityAssertion ();

peerConnection.onpeeridentity = function(e) {
  console.log("IdP= " + e.target.peerIdentity.idp +
              " identity=" + e.target.peerIdentity.name);
};
```

Happens behind the scenes

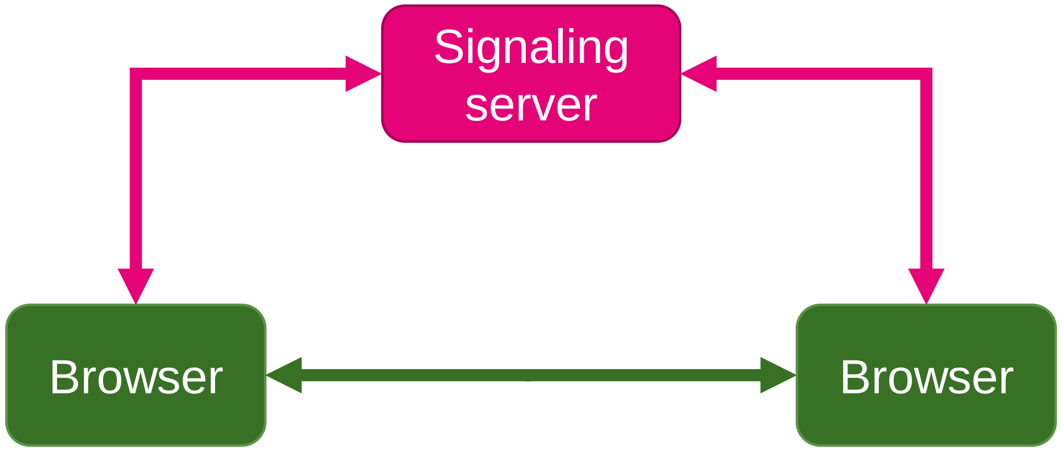
DistriNet iMinds KU LEUVEN STREWS

WebRTC deployments

Various WebRTC deployments

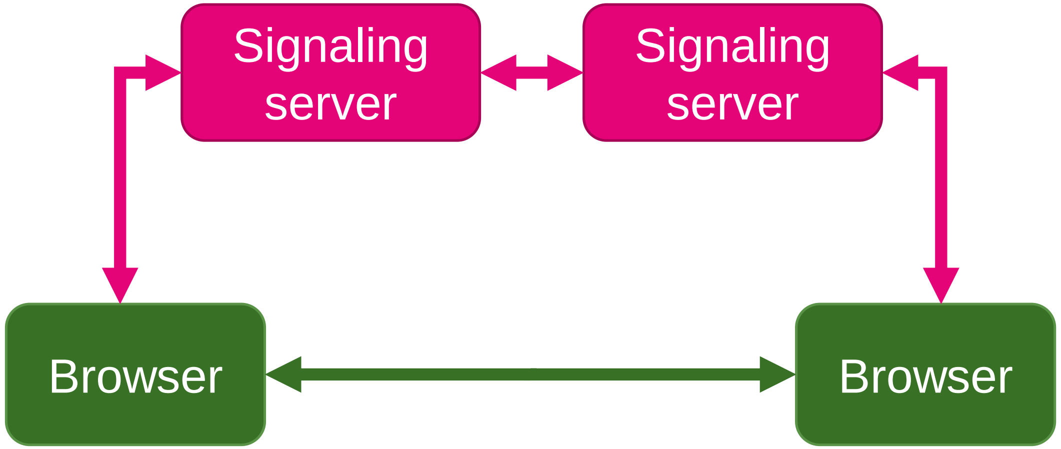


Helpdesk call

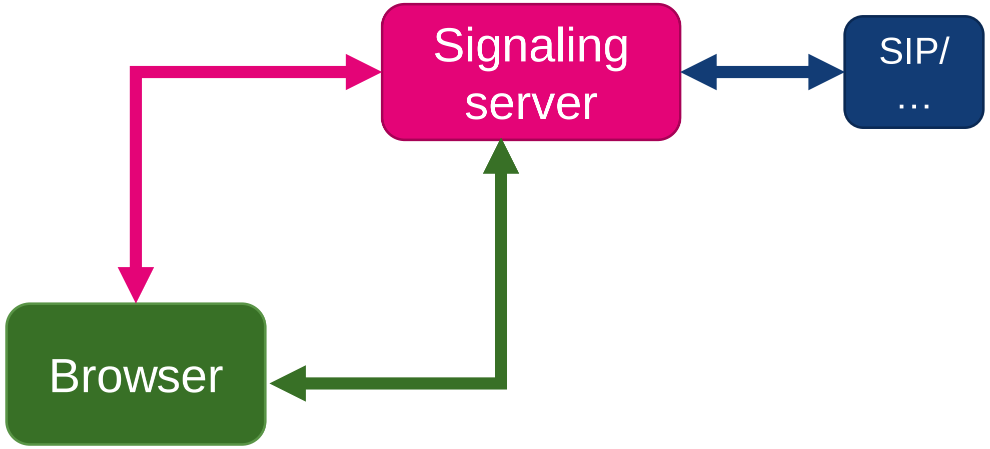


2-party video conferencing

Federated signaling setup

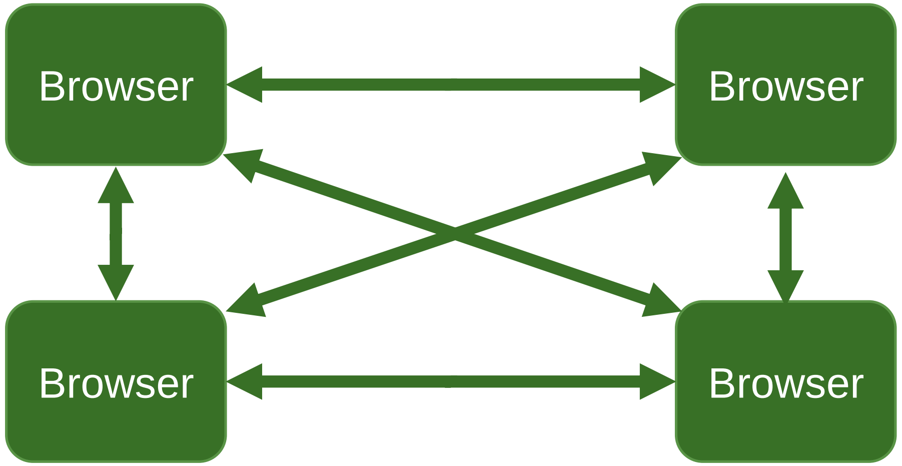


Bridged to SIP/Jingle/… infrastructure



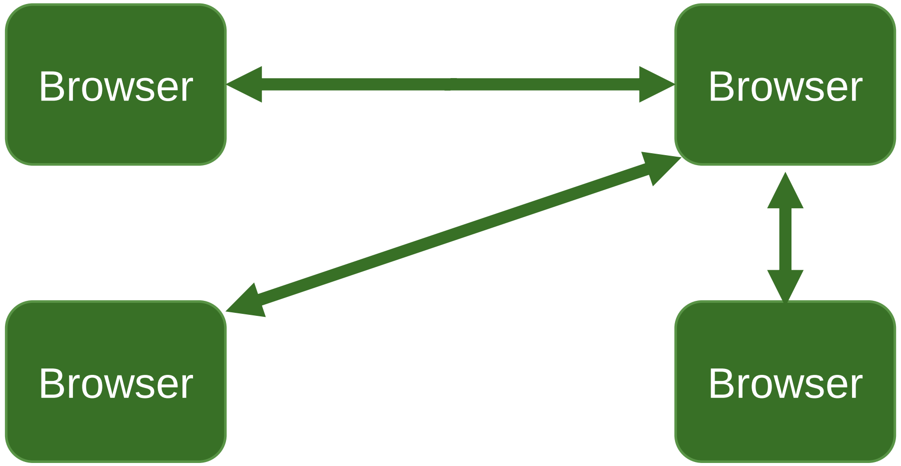
Multiple peer topologies

Peer to Peer connection

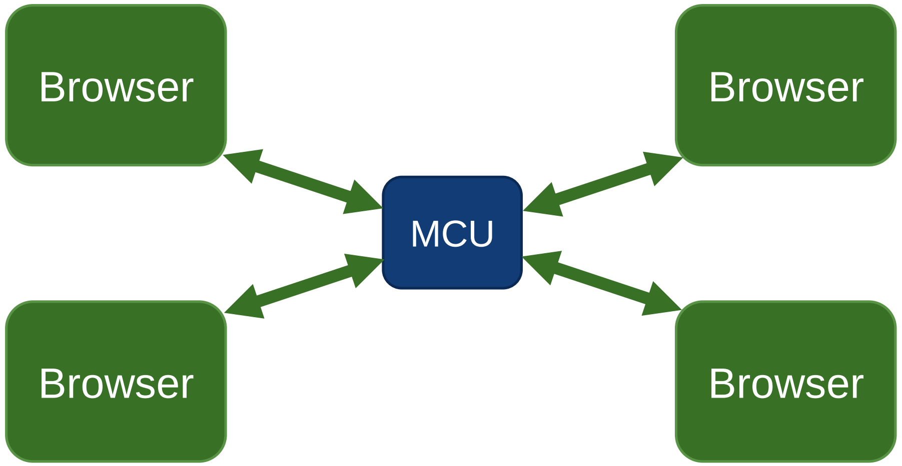


Mesh network

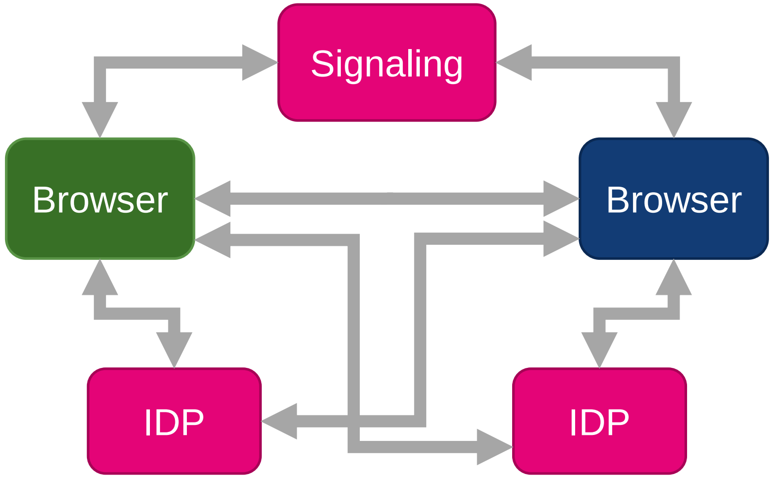
Star network



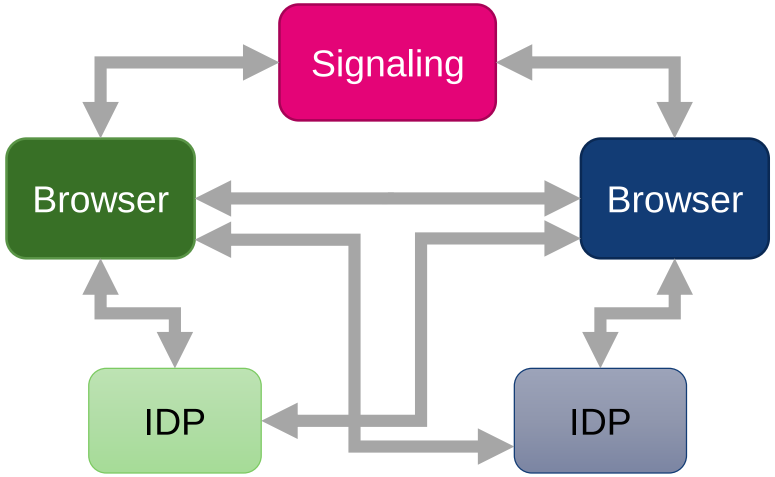
Multipoint Control Unit (MCU)



IDP setups

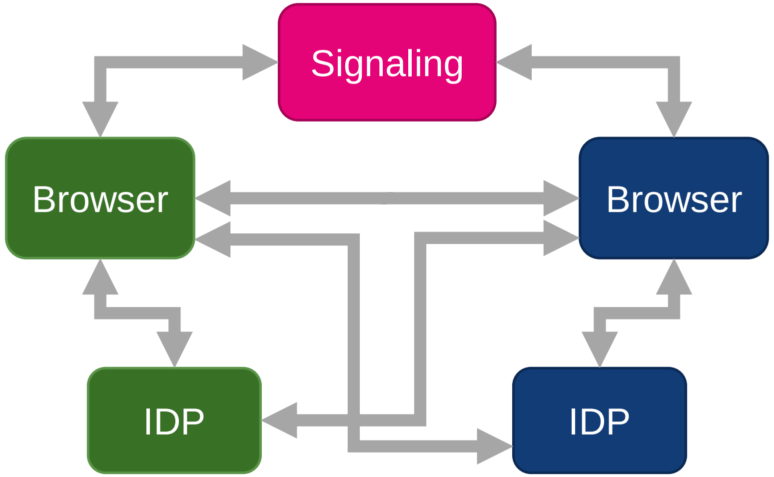


IDP = Signaling Server

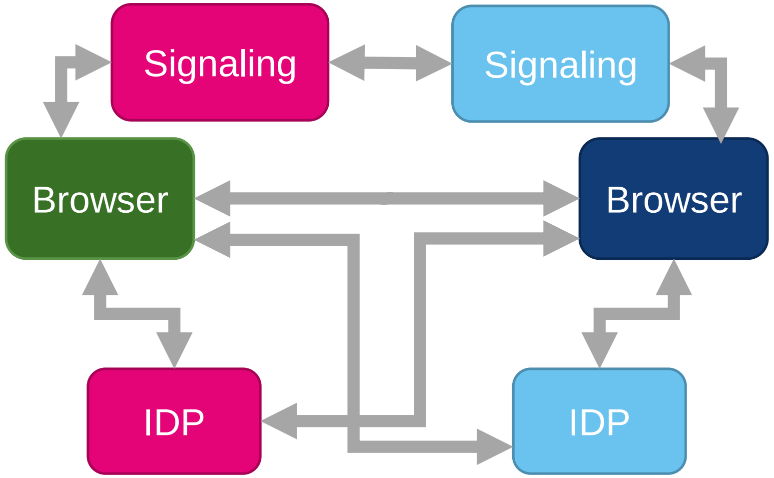


User chooses own IDP

Browser chooses IDP



Signaling server chooses IDP



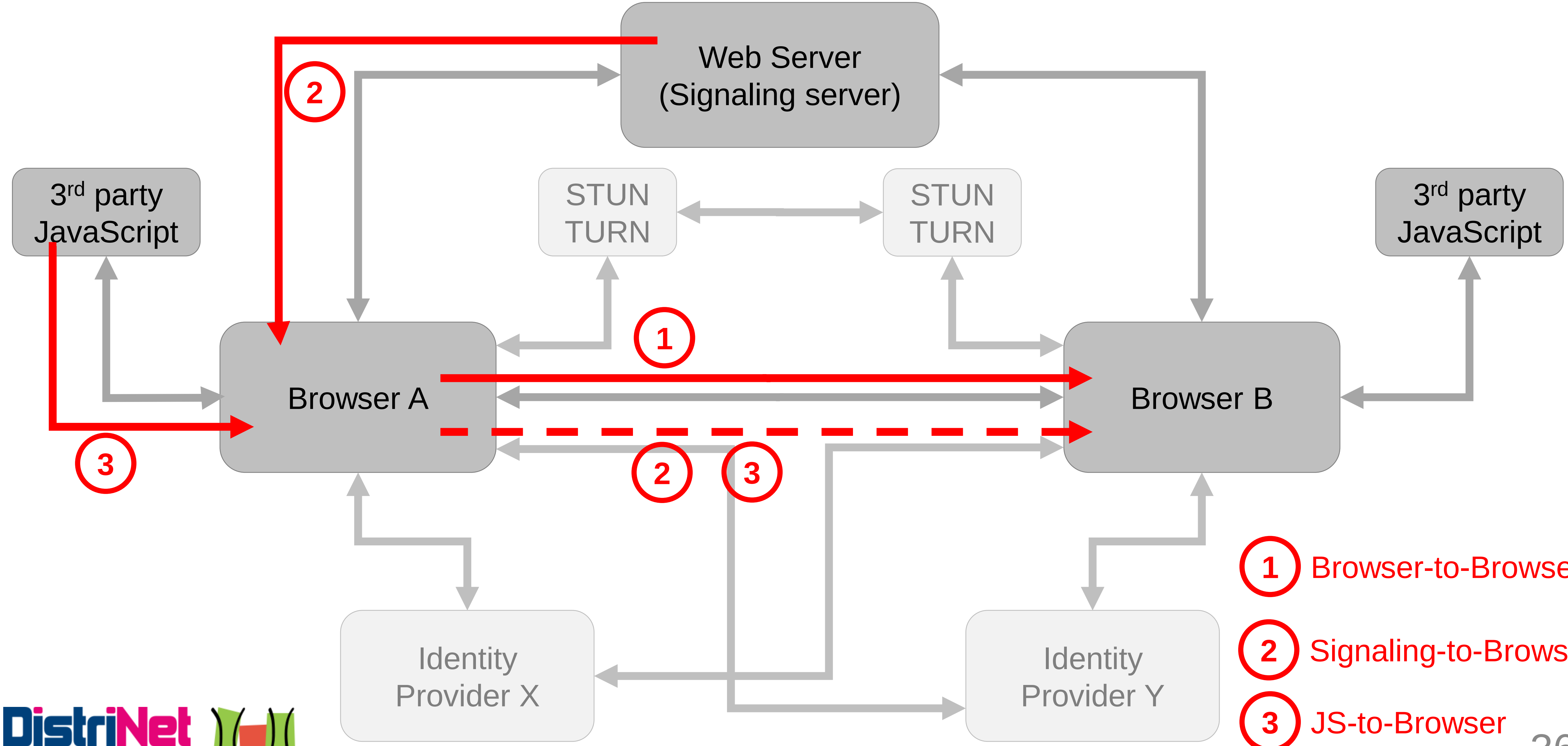
Overview of attack vectors

Attack overview

- Classical web attacks still apply
- WebRTC permission model
- Potential confidentiality leaks
- Endpoint authenticity
- Attacking the underlying infrastructure

#1 Classical web attacks still apply

Client-side web attacks still apply



1 Browser-to-Browser

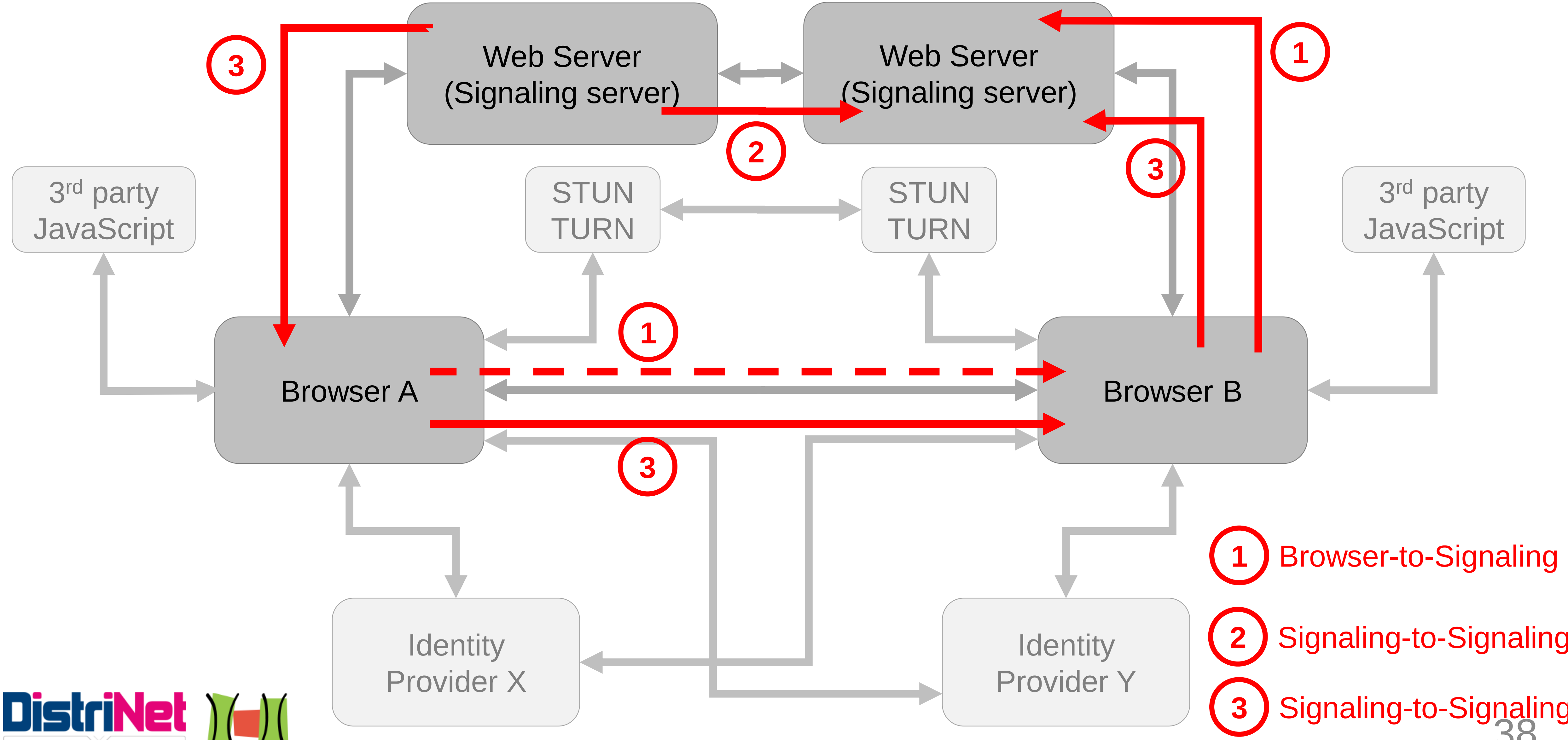
2 Signaling-to-Browser

3 JS-to-Browser

Classical attacks in WebRTC setup

- Attacks such as XSS also apply in a WebRTC setup
  - New attack surface added
  - New capabilities can be achieved
  - Hard to trace back to its origin
- Attacks can cross origins and browsers, via peer-to-peer connection

Don’t forget the server-side web attacks



#2 WebRTC permission model

Permission model

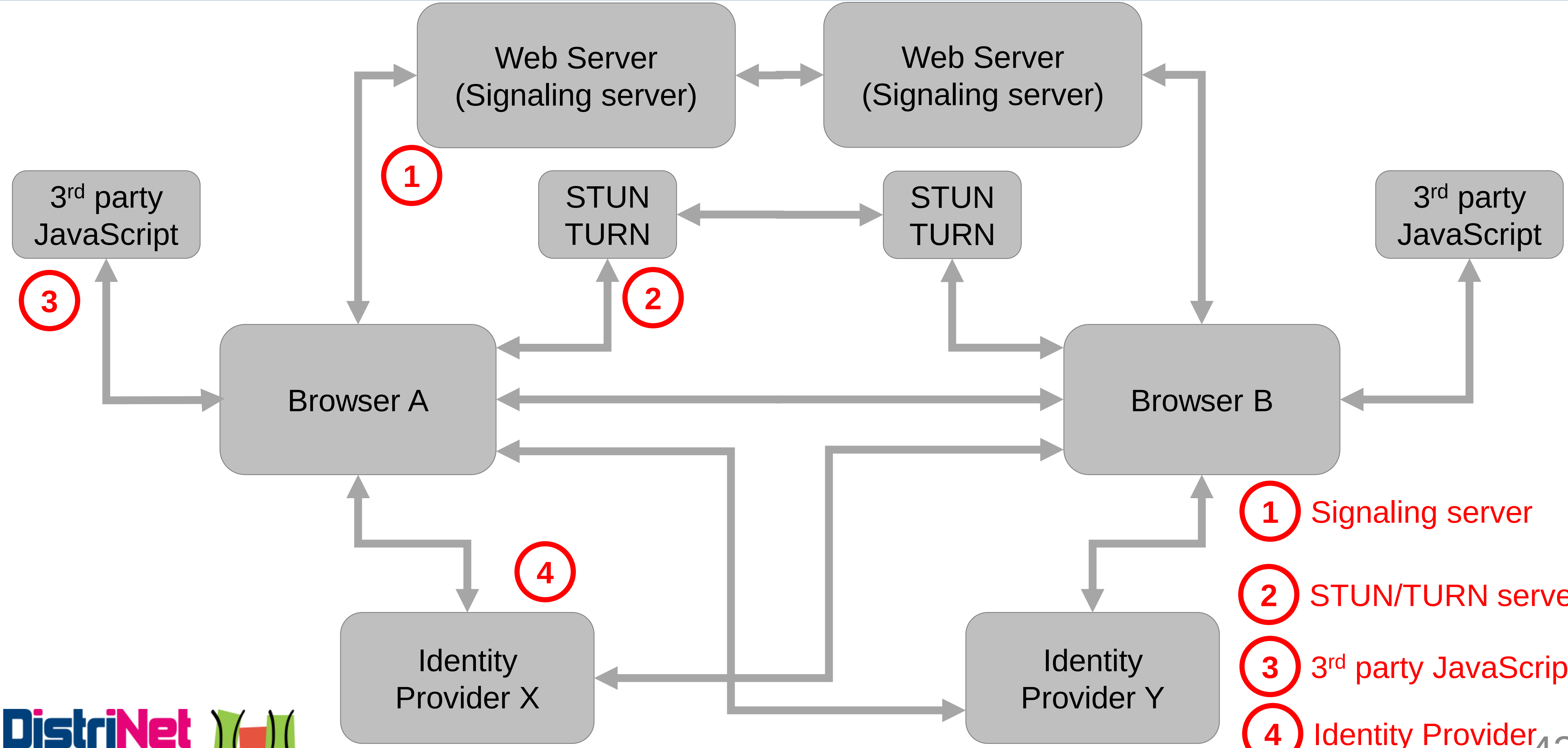
- For which operation, user consent is required?
  - Camera? ✓
  - Microphone? ✓
  - Getting network characteristics (ICE)? ✘
  - Setting up a peer-to-peer connection? ✘
  - Sending your audio/video to a remote peer? ✘
  - Sharing your screen? ✘ ✓
  - Selecting an appropriate Identity Provider? ✘
  - Veryfying your endpoint's identity? ✘

Potential issues due to permission model

- Multiple streams of your camera been sent to different parties
- Phishing opportunities for IdP authentication
- ICE fingerprinting
- Screen sharing via extension
- Verification of endpoint's authenticity depends on:
  - Signaling server setting up IdP authentication and verification
  - Browser setting up selection of IdP
  - Browser displaying IdP verification

#3 Potential confidentiality leaks

Meta-data leakage: Trace that communication has happened



1 Signaling server

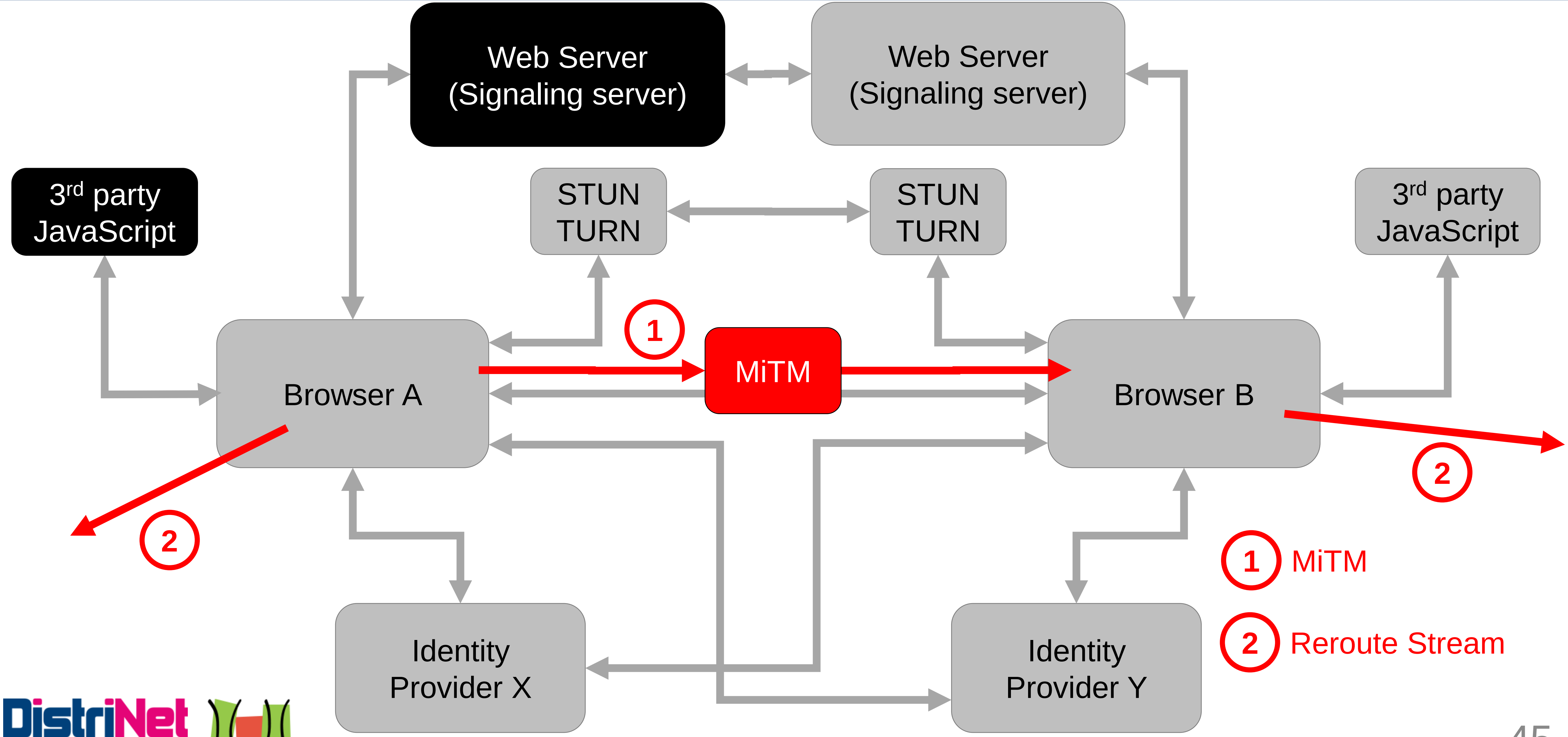
2 STUN/TURN server

3 3rd party JavaScript

4 Identity Provider

DistriNet iMinds KU LEUVEN STREWS

Eavesdropping on the connection

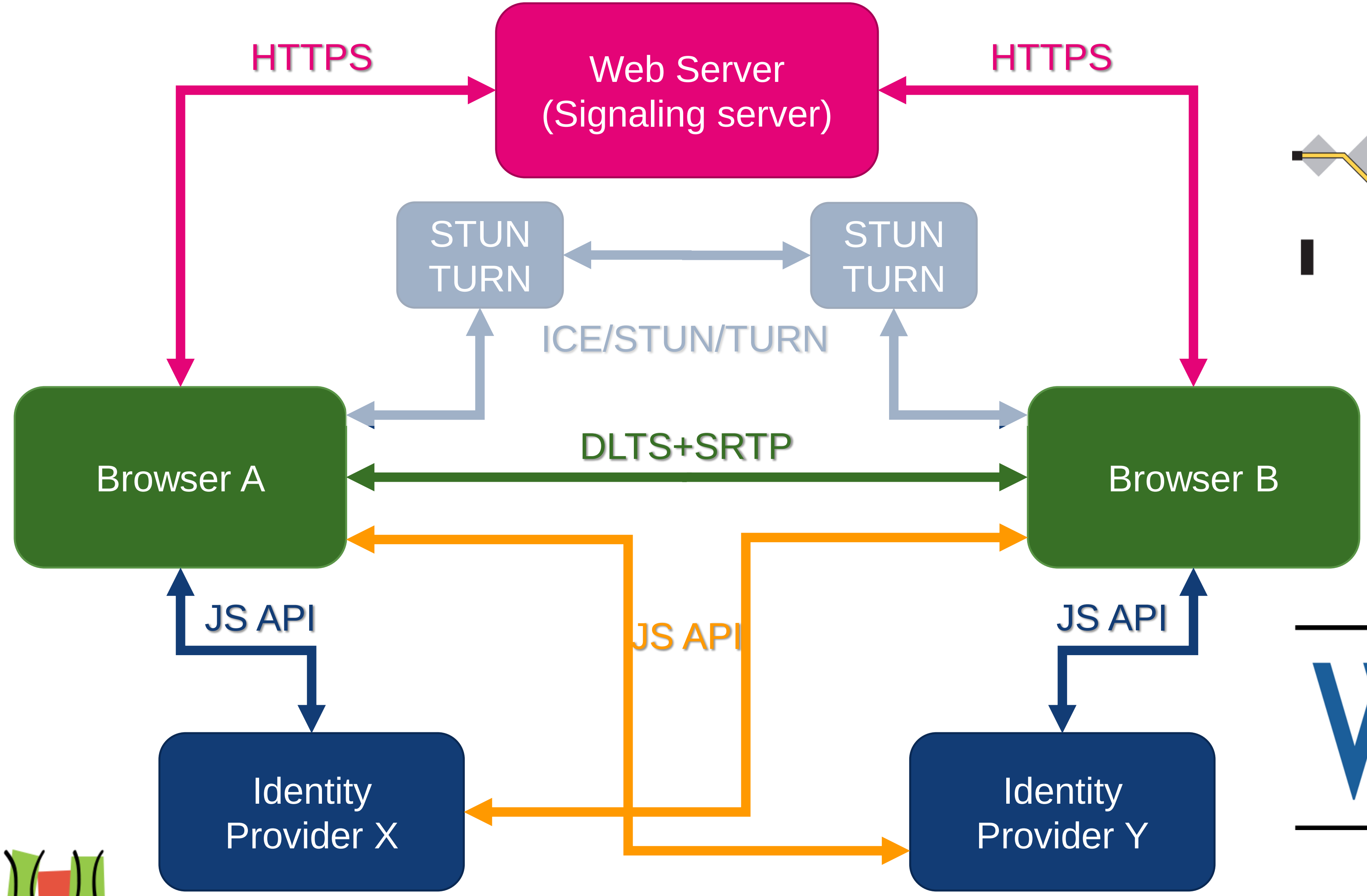


1 MiTM

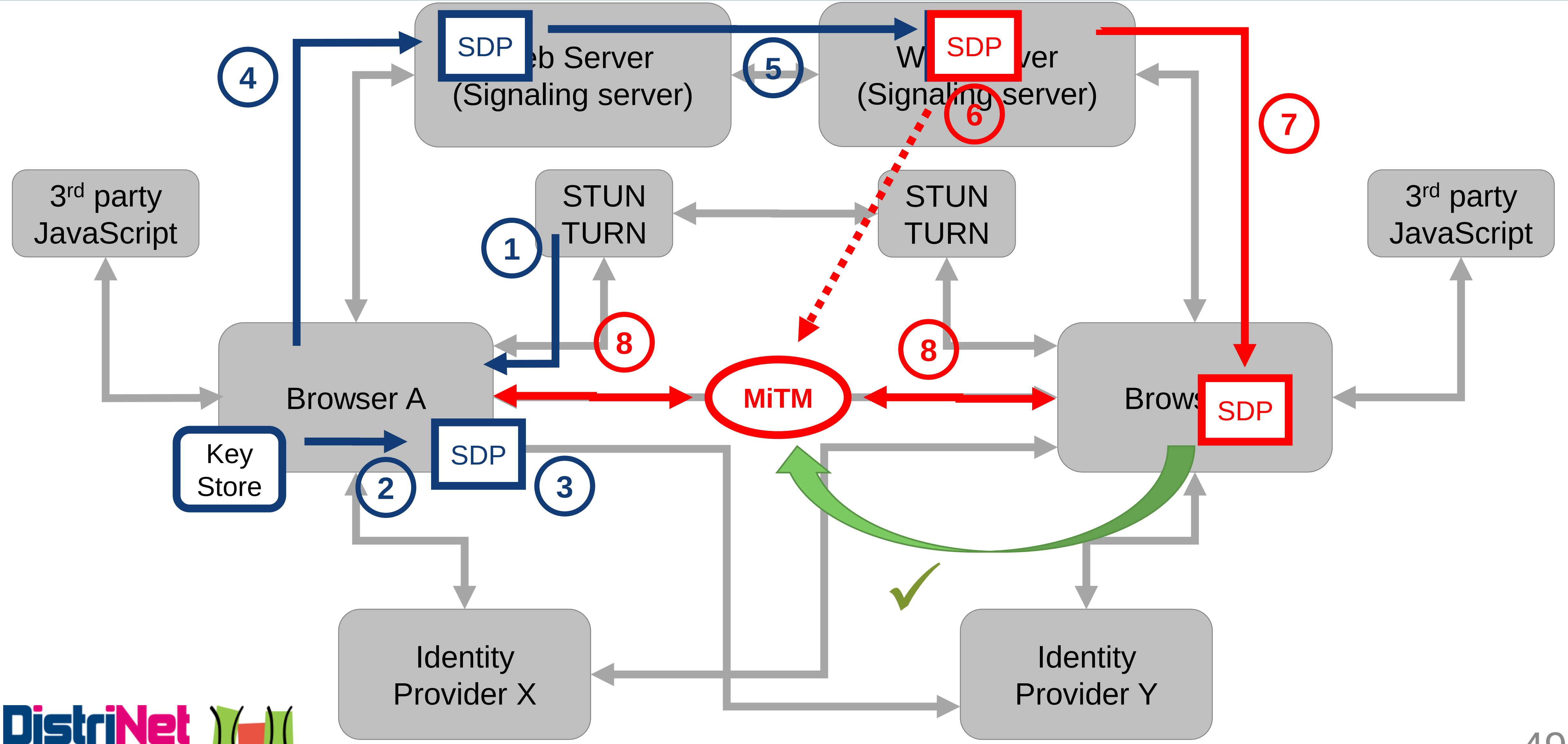
2 Reroute Stream

#4 Endpoint authenticity

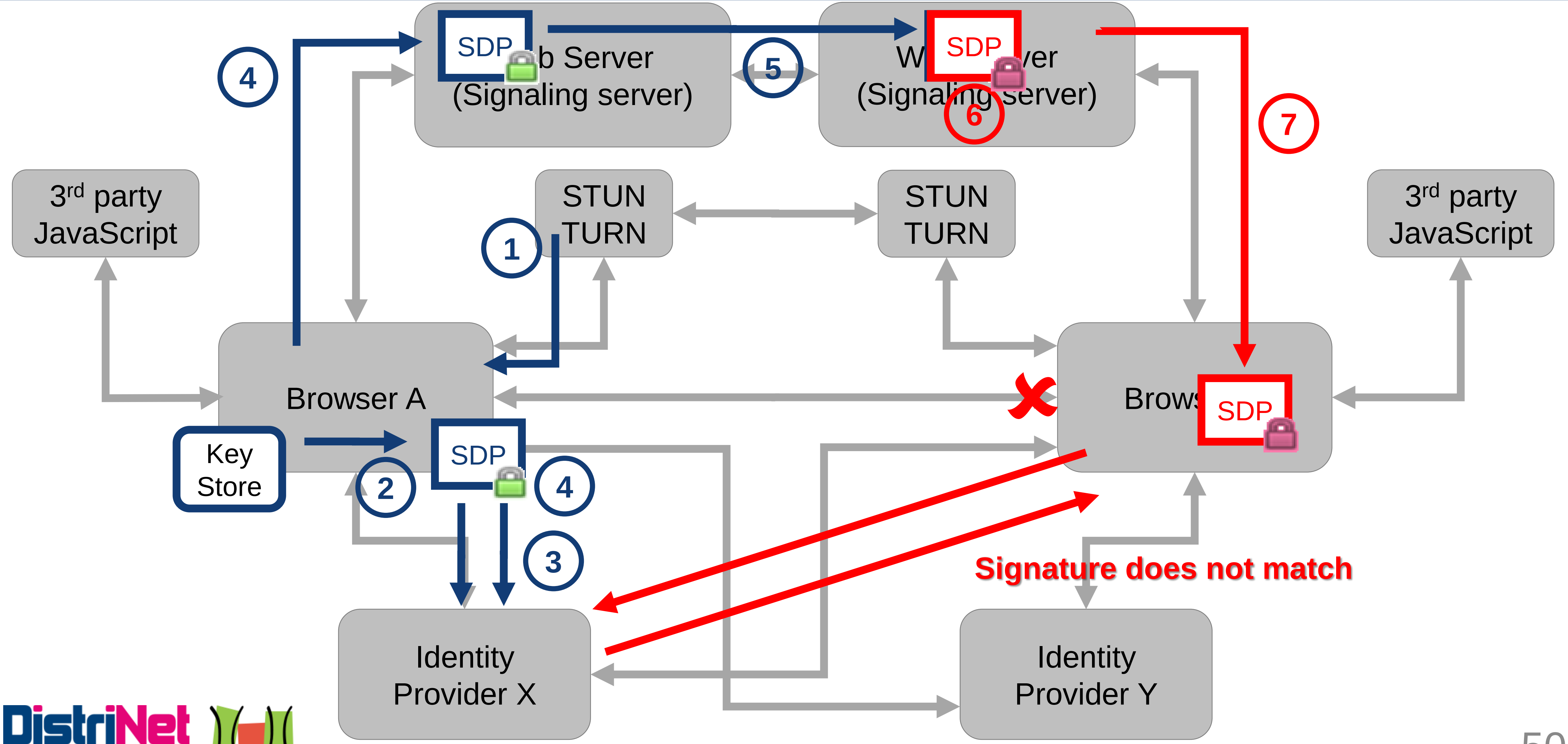
WebRTC architecture



Setting up the media channel without IdP



Setting up the media channel with the IdP

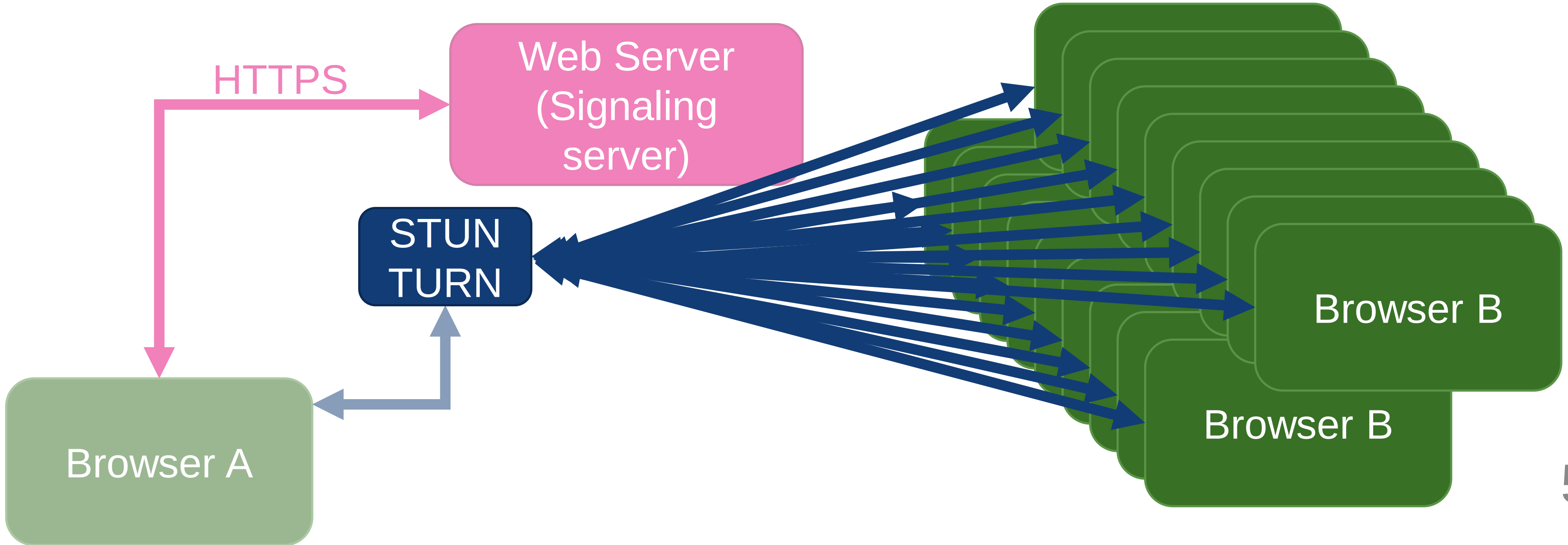


DistriNet iMinds KU LEUVEN STREWS

#5 Attacking the underlying infrastructure

There are many other things to consider…

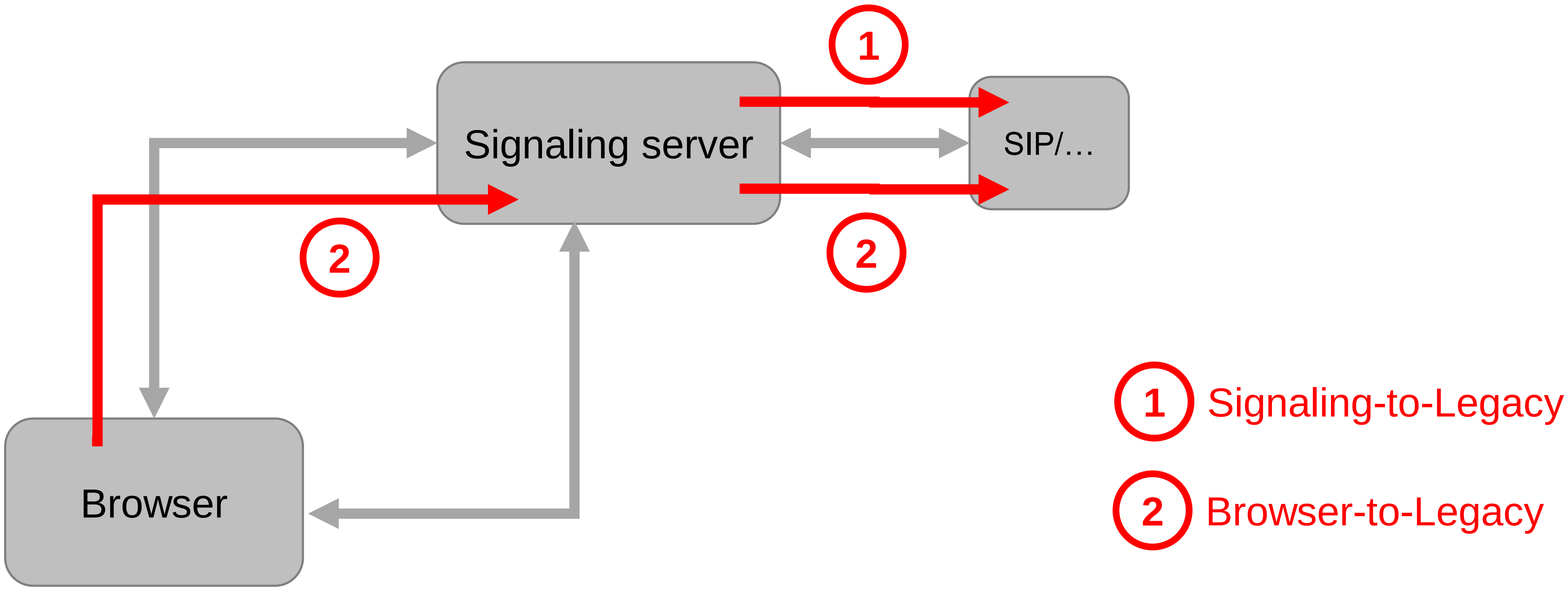
- STUN/TURN server share symmetric secret for authentication
  - Steal bandwidth of other tenants
  - Trigger DDoS attacks on/via STUN/TURN



DistriNet iMinds KU LEUVEN STREWS

There are many other things to consider...

- Attack legacy components such as SIP/VOIP/Jingle/…



Wrap-up

Flexibility vs security?

- Many deployment variation make it hard to assess the *end-to-end* security

- Very *open* specifications
  - Signaling path is not specified
  - Limited specification on how to interact with the IdP
  - Responsibility for IdP settings is unspecified
    - Could be the responsibility of the end-user, browser, signaling server
    - No obligation to use IdPs (i.e. unauthenticated endpoints by default)
    - Unspecified how identity management needs to be visualized

DistriNet iMinds KU LEUVEN STREWS

Minimal permission model enforced

- Only permission for capture of video/audio is currently needed
- A PeerConnection can be setup without user intervention
  - A user does not know on what connection a stream is used
  - Streams can be cloned over multiple connections
- PeerConnections can be set up without authenticating endpoints
- Websites can trigger ICE fingerprints

WebRTC: security impact on applications?

- Even if your application is not using WebRTC, new attack vectors and attacker capabilities may apply
  - Browser-to-browser attacks, …
  - ICE fingerprints
  - Screen sharing to circumvent other security mechanisms
  - …

Peer-to-peer provides more privacy?

- Peer-to-peer could provide more confidentiality in user communication

- But:
  - Possible collection of meta-data at signaling server, STUN/TURN server, IdP provider, …
  - Possible confidentiality leaks in communication channel
    - MiTM attacks in unauthenticated endpoints
    - Streams being cloned and rerouted

Taking home message

- Limit trust in third-party libraries running in your origin
  - If possible, isolate the WebRTC functionality in a separate origin (e.g. subdomain)
- Use an identity provider to authenticate the end-points
- Use best-practices in protecting your application
  - Don’t forget the browser-to-browser, signaling-to-signaling communication
- Be careful with screen sharing extension
- Embrace the new browser capabilities!

Real-Time communication with WebRTC

Lieven Desmet – iMinds-DistriNet, KU Leuven
Lieven.Desmet@cs.kuleuven.be

SecAppDev Leuven 2015 (27/02/2015, Leuven)