WEBRTC IN C++

- Yuri Valentini
- Italian C++ Day
  28 Novembre 2020

SOMMARIO

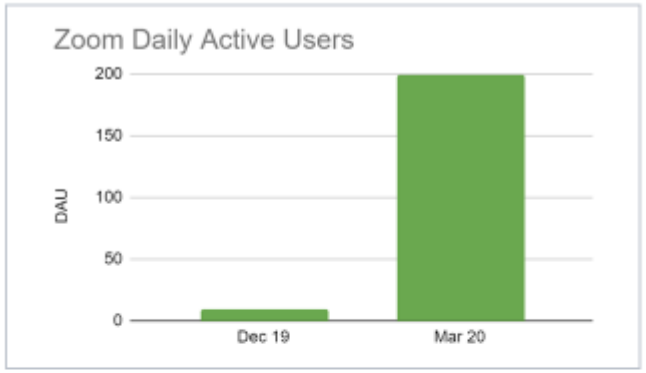
- Introduzione a WebRTC
- Concetti base VOIP
- Comunicazione peer-to-peer
- C++ in stile webrtc
- Esempio DataChannel
- Architettura videoconferenza

YURI VALENTINI

- Sviluppo Windows e Linux
- C/C++, C#, Python
- Videoconferenza e VOIP
- yuroller@gmail.com
- https://github.com/yuroller

- Web Real-Time Communication
- browser, mobile-phones, IoT
- BSD license
- No brevetti

CRESCITA VIDEOCONFERENZA



Google Meet

- **3M** new users a day
- **100M** daily participants

Microsoft Teams

- **44M** DAU in March
- **75M** DAU in April

Cisco WebEx

- **324M** participants in March
- **14B** meeting minutes in March
- **20B** meeting minutes in April

BlogGeek.Me

da bloggeek.me

STORIA

- 2010 Google acquisisce GIPS ($68.2 mil)
- 2011 Google rilascia WebRTC
- IETF standard protocolli
- W3C standard api browser
- 2013 video call fra browser diversi
- 2014 OpenWebRTC di Ericsson Research
- 2017 WebRTC 1.0 Candidate Recommendation
- W3C Editor's Draft 12 November 2020

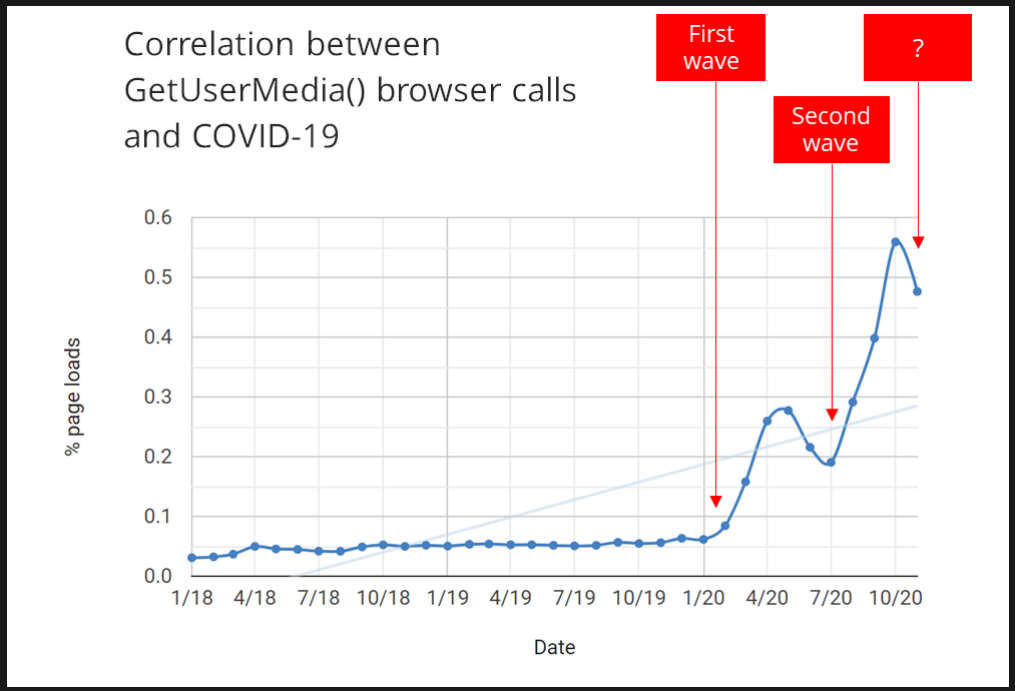
SUPPORTO

- PC (Chrome, Firefox, Edge, Safari)
- Android (Chrome, Firefox)
- IOS 11+ (MobileSafari/WebKit)
- Chromecast (Google Stadia)
- ...

API JAVASCRIPT

- getUserMedia: accesso ai dispositivi
- RTCPeerConnection: audio/video tra peer
  - elaborazione segnali (aec, noise reduction, ...)
  - audio/video codec
  - comunicazione fra peer
  - sicurezza (DTLS, SRTP)
  - gestione banda (TWCC, ...)
- RTCDataChannel: trasferimento dati
- getStats: ottenere statistiche

UTILIZZO API GETUSERMEDIA



da bloggeek.me

AUDIO VIDEO CODEC

- obbligatori
  - PCMA/PCMU, Opus
  - DTMF via Telephone Event
- altri di Google WebRTC
  - ISAC, G722, ILBC
  - CN (Confort Noise)
  - VP8, VP9
- esterni: es. H264, AV1

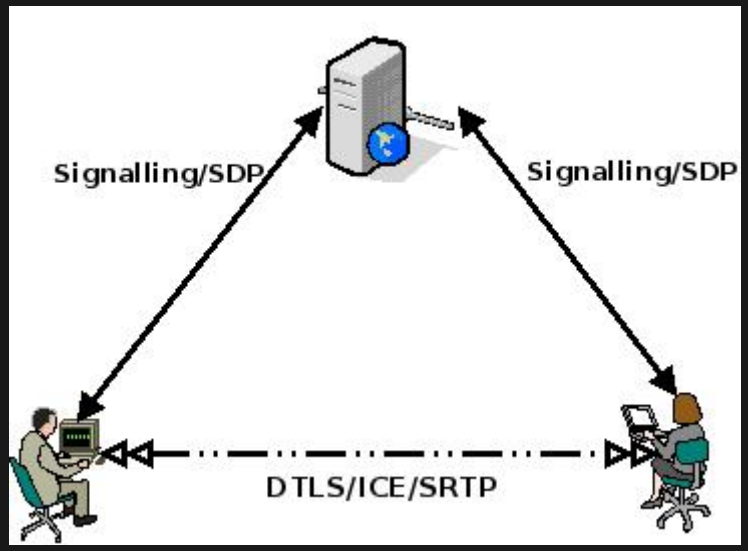
SDP

- descrive i media
- negoziazione offerta/risposta
- utilizzato in VOIP SIP
- sdp anatomy
- "alternativa" ORTC (DEAD)

TERMINOLOGIA VOIP/WEBRTC

- RTP, RTCP, SCTP
- DTLS, SRTP
- STUN (Session Traversal Utilities for NAT)
- TURN (Traversal Using Relays around NAT)
- ICE (Interactive Connectivity Establishment)

PEER-TO-PEER



da webrtchacks.com

INSTALLAZIONE SU WINDOWS

scaricare e scompattare depot_tools

```
c:\progetti>PATH=C:\progetti\depot_tools;%PATH%
c:\progetti>set DEPOT_TOOLS_WIN_TOOLCHAIN=0
c:\progetti>gclient
c:\progetti\webrtc>fetch webrtc
c:\progetti\webrtc\src>gclient sync
c:\progetti\webrtc\src>gn gen out\Release --ide=vs \
  --args="is_clang=false is_debug=false rtc_include_tests=false"
c:\progetti\webrtc\src>ninja -C out\Release
```

INSTALLAZIONE SU LINUX

```
~/progetti$ git clone \
  https://chromium.googlesource.com/chromium/tools/depot_tools.git
~/progetti$ export PATH=$HOME/progetti/depot_tools:$PATH
~/progetti$ gclient
~/progetti/webrtc$ fetch webrtc
~/progetti/webrtc$ gclient sync
~/progetti/webrtc$ gn gen out\Release \
  --args="is_debug=false rtc_include_tests=false"
~/progetti/webrtc$ ninja -C out\Release
```

#INCLUDE

- Interfaccia "stabile" (api/* rtc_base/*)
- Interfaccia privata (ad es. modules/*):
  - modules/audio_coding
  - modules/video_coding
  - modules/desktop_capture
  - modules/audio_device
  - modules/audio_mixer
  - modules/rtp_rtcp
  - ...

GOOGLE C++

- C++11 (abseil)
- exception sono vietate
- uso limitato degli stream stl
- template "semplici"

RTC::THREAD

- implementa rtc::TaskQueueBase
- inviare e ricevere messaggi (Post(), Get(), ..)
- eseguire codice su thread (Invoke<T>())
- socket server (Thread::CreateWithSocketServer())

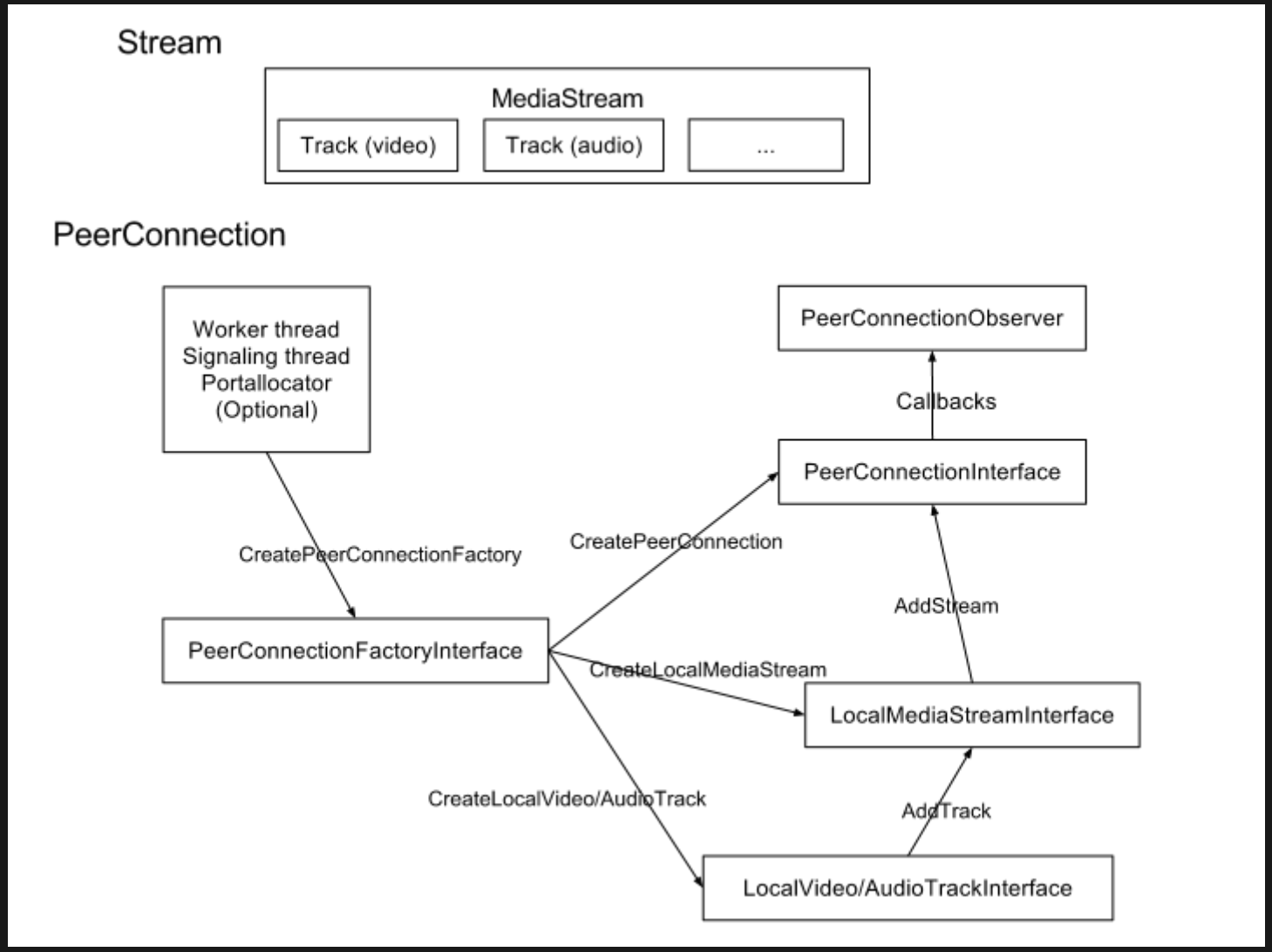
RTC::SCOPED_REFPTR

- reference count interno
- tracciabilità di oggetti reference counted (es. COM)
- stessa memoria fra T e rtc::scoped_refptr<T>
- meno allocazioni/deallocazioni
- località di memoria

OBSERVER WEBRTC

- esito di operazione asincrone
- eventi asincroni
- eseguito da thread specifico
- gestione memoria semplificata

OGGETTI WEBRTC



PEERCONNECTION 1

```
class PeerConnectionInterface : public rtc::RefCountInterface
public:
  virtual rtc::scoped_refptr<StreamCollectionInterface>
    local_streams() = 0;
  virtual rtc::scoped_refptr<StreamCollectionInterface>
    remote_streams() = 0;
  virtual bool AddStream(MediaStreamInterface* stream) = 0;
  virtual void RemoveStream(MediaStreamInterface* stream) = 0;
  virtual RTCErrorOr<rtc::scoped_refptr<RtpSenderInterface>>
    AddTrack(
      rtc::scoped_refptr<MediaStreamTrackInterface> track,
      const std::vector<std::string>& stream_ids) = 0;
  virtual bool RemoveTrack(RtpSenderInterface* sender) = 0;
...
```

PEERCONNECTION 2

```
...
  virtual RTCErrorOr<rtc::scoped_refptr<RtpTransceiverInterfac
    AddTransceiver(rtc::scoped_refptr<MediaStreamTrackInterfac
      track) = 0;
  virtual RTCErrorOr<rtc::scoped_refptr<RtpTransceiverInterfac
    AddTransceiver(rtc::scoped_refptr<MediaStreamTrackInterfac
          track, const RtpTransceiverInit& init) = 0;

  virtual std::vector<rtc::scoped_refptr<RtpTransceiverInterfa
    GetTransceivers() const = 0;
...
```

PEERCONNECTION 3

```
...
  virtual void GetStats(
    rtc::scoped_refptr<RtpSenderInterface> selector,
    rtc::scoped_refptr<RTCStatsCollectorCallback> callback) {}
  virtual void GetStats(
    rtc::scoped_refptr<RtpReceiverInterface> selector,
    rtc::scoped_refptr<RTCStatsCollectorCallback> callback) {}

  virtual rtc::scoped_refptr<DataChannelInterface>
    CreateDataChannel(
      const std::string& label,
      const DataChannelInit* config) = 0;
...
```

PEERCONNECTION 4

```
...
  virtual void CreateOffer(
    CreateSessionDescriptionObserver* observer,
    const RTCOfferAnswerOptions& options) = 0;
  virtual void CreateAnswer(
    CreateSessionDescriptionObserver* observer,
    const RTCOfferAnswerOptions& options) = 0;
  virtual void SetLocalDescription(
    SetSessionDescriptionObserver* observer,
    SessionDescriptionInterface* desc) = 0;
  virtual void SetRemoteDescription(
    SetSessionDescriptionObserver* observer,
    SessionDescriptionInterface* desc) {}
...
```

PEERCONNECTION 5

```
...
  virtual bool AddIceCandidate(
    const IceCandidateInterface* candidate) = 0;
  virtual void Close() = 0;
};
```

PEERCONNECTIONOBSERVER 1

```
class PeerConnectionObserver {
public:
  virtual ~PeerConnectionObserver() = default;

  virtual void OnSignalingChange(
    PeerConnectionInterface::SignalingState new_state) = 0;

  virtual void OnAddStream(
    rtc::scoped_refptr<MediaStreamInterface> stream) {}
  virtual void OnRemoveStream(
    rtc::scoped_refptr<MediaStreamInterface> stream) {}
  virtual void OnDataChannel(
    rtc::scoped_refptr<DataChannelInterface> data_channel) = 0
...
```

PEERCONNECTIONOBSERVER 2

```
...
  virtual void OnRenegotiationNeeded() = 0;
  virtual void OnIceConnectionChange(
      PeerConnectionInterface::IceConnectionState new_state) {
  virtual void OnConnectionChange(
      PeerConnectionInterface::PeerConnectionState new_state)
  virtual void OnIceGatheringChange(
      PeerConnectionInterface::IceGatheringState new_state) =
  virtual void OnIceCandidate(
    const IceCandidateInterface* candidate) = 0;
...
```

PEERCONNECTIONOBSERVER 3

```
...
  virtual void OnAddTrack(
    rtc::scoped_refptr<RtpReceiverInterface> receiver,
    const std::vector<rtc::scoped_refptr<MediaStreamInterface>
      streams) {}
  virtual void OnTrack(
    rtc::scoped_refptr<RtpTransceiverInterface> transceiver) {
  virtual void OnRemoveTrack(
    rtc::scoped_refptr<RtpReceiverInterface> receiver) {}
};
```

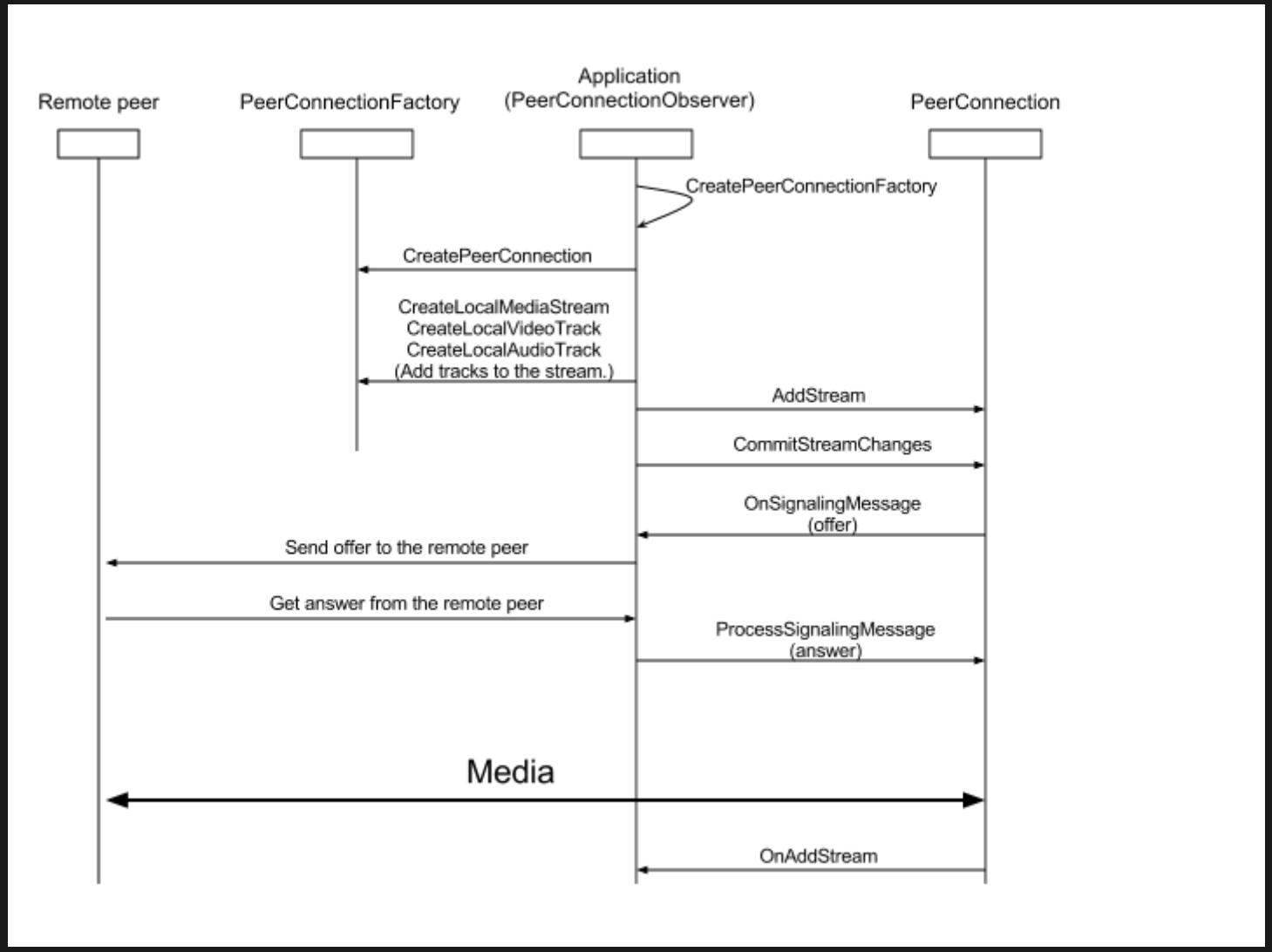
MEDIASTREAM 1

```
class MediaStreamInterface : public rtc::RefCountInterface,
    public NotifierInterface {
public:
  virtual std::string id() const = 0;

  virtual AudioTrackVector GetAudioTracks() = 0;
  virtual VideoTrackVector GetVideoTracks() = 0;
  virtual rtc::scoped_refptr<AudioTrackInterface> FindAudioTra
    const std::string& track_id) = 0;
  virtual rtc::scoped_refptr<VideoTrackInterface> FindVideoTra
        const std::string& track_id) = 0;
...
```

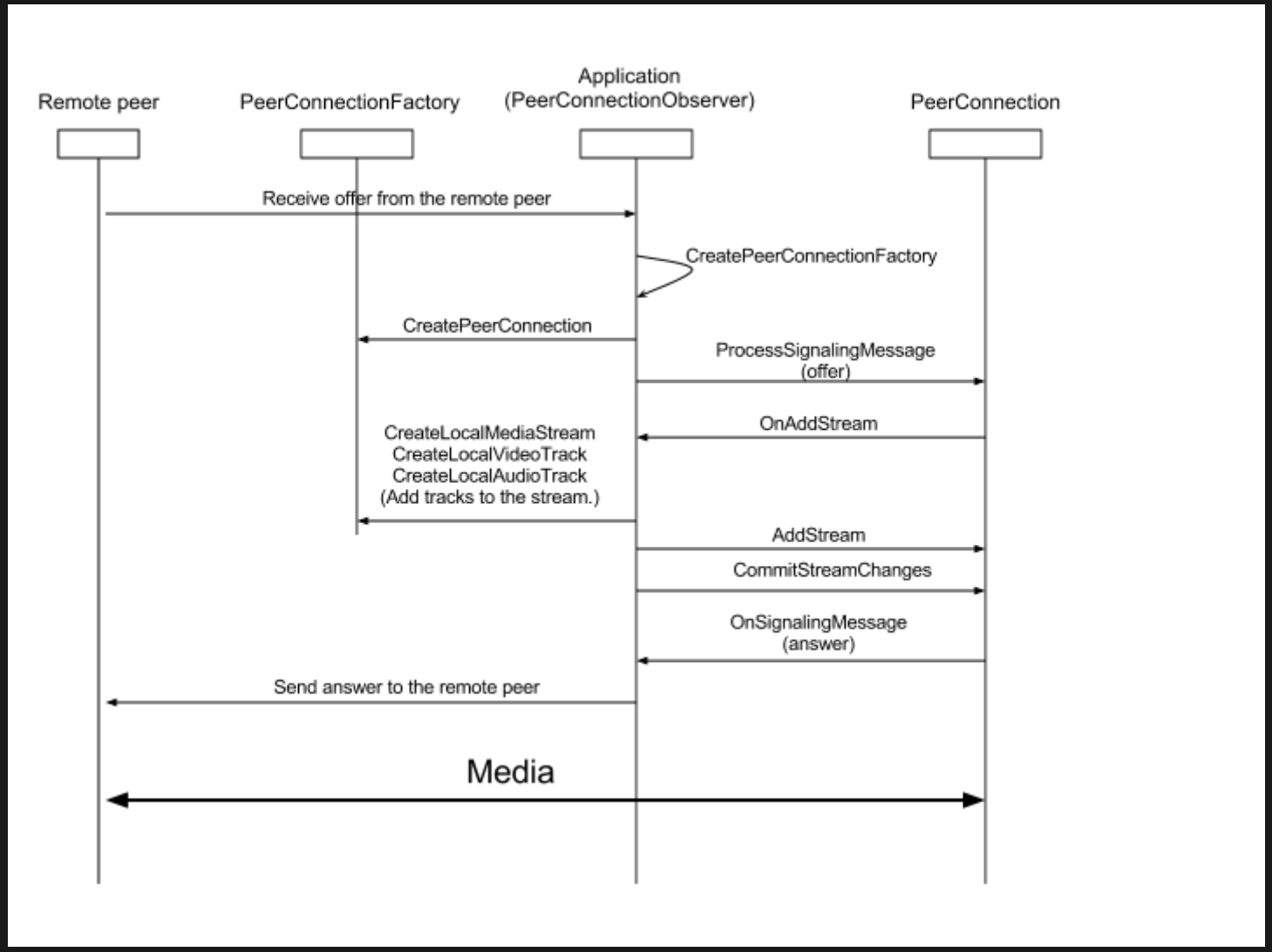
MEDIASTREAM 2

```
...
  virtual bool AddTrack(AudioTrackInterface* track) = 0;
  virtual bool AddTrack(VideoTrackInterface* track) = 0;
  virtual bool RemoveTrack(AudioTrackInterface* track) = 0;
  virtual bool RemoveTrack(VideoTrackInterface* track) = 0;

protected:
  ~MediaStreamInterface() override = default;
};
```

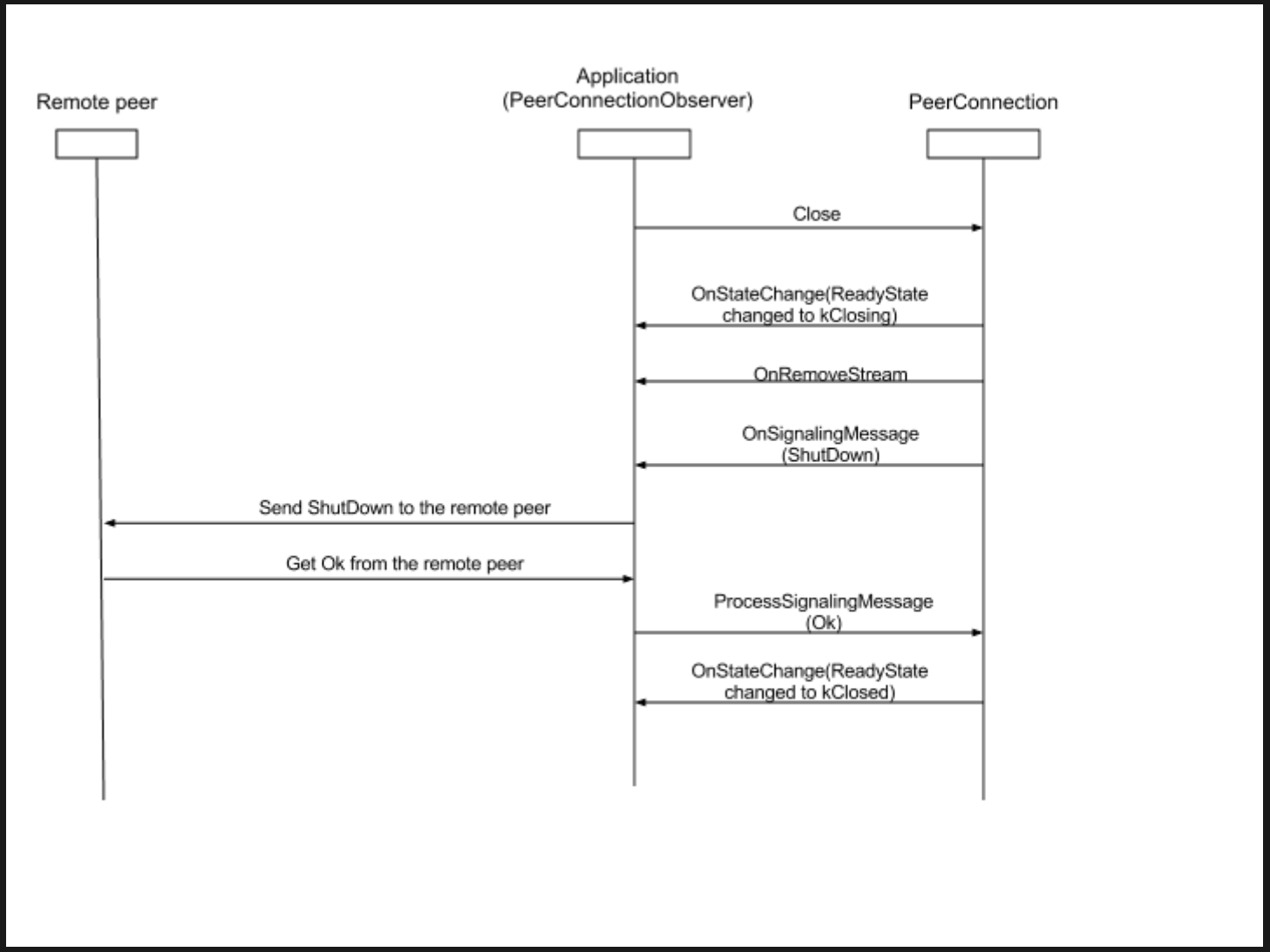
FARE UNA CHIAMATA



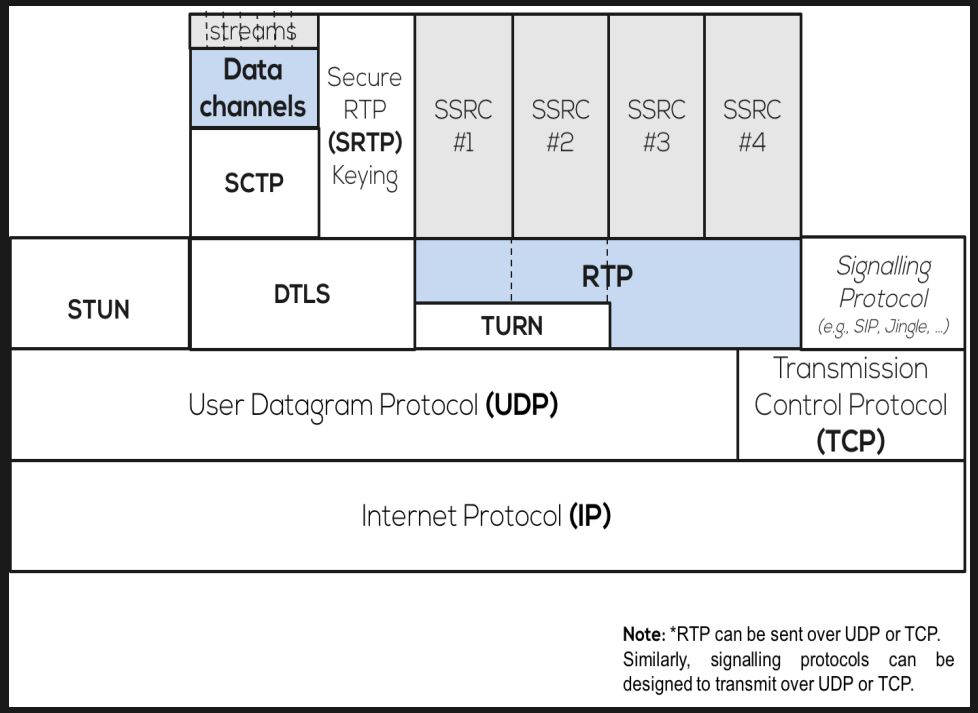
RICEVERE UNA CHIAMATA



TERMINARE UNA CHIAMATA



PROTOCOLLI

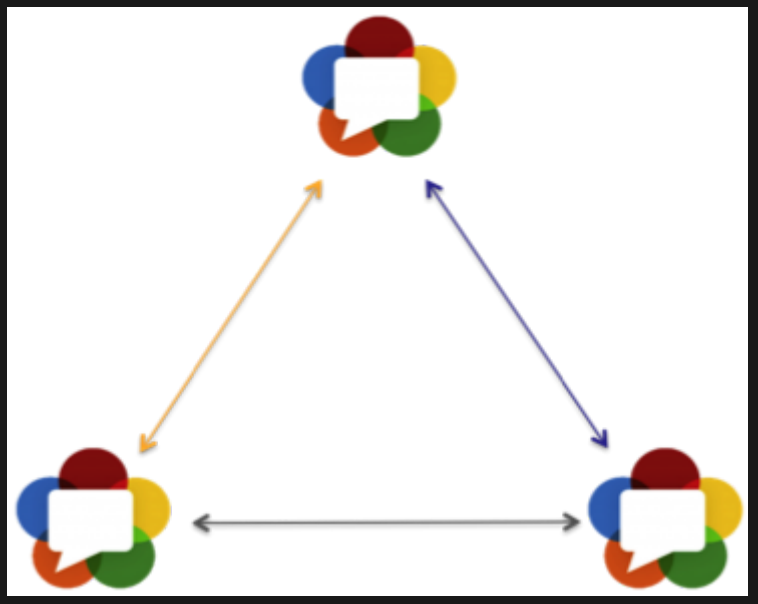


da callstats.io

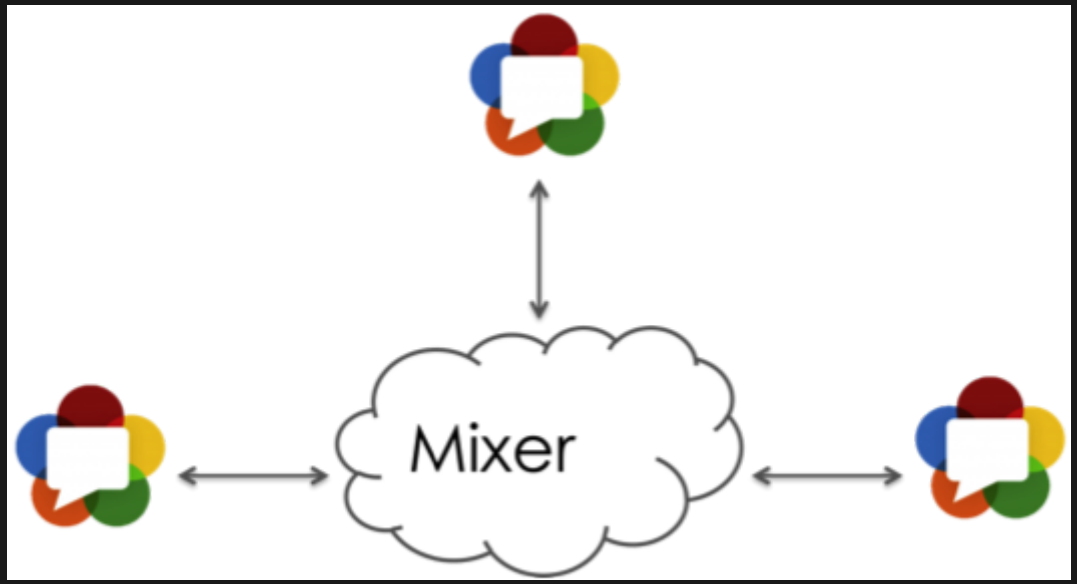
VIDEOCONFERENZA

- Mesh (Peer-To-Peer)
- Mixer (MCU)
- Router (SFU)

PEER-TO-PEER

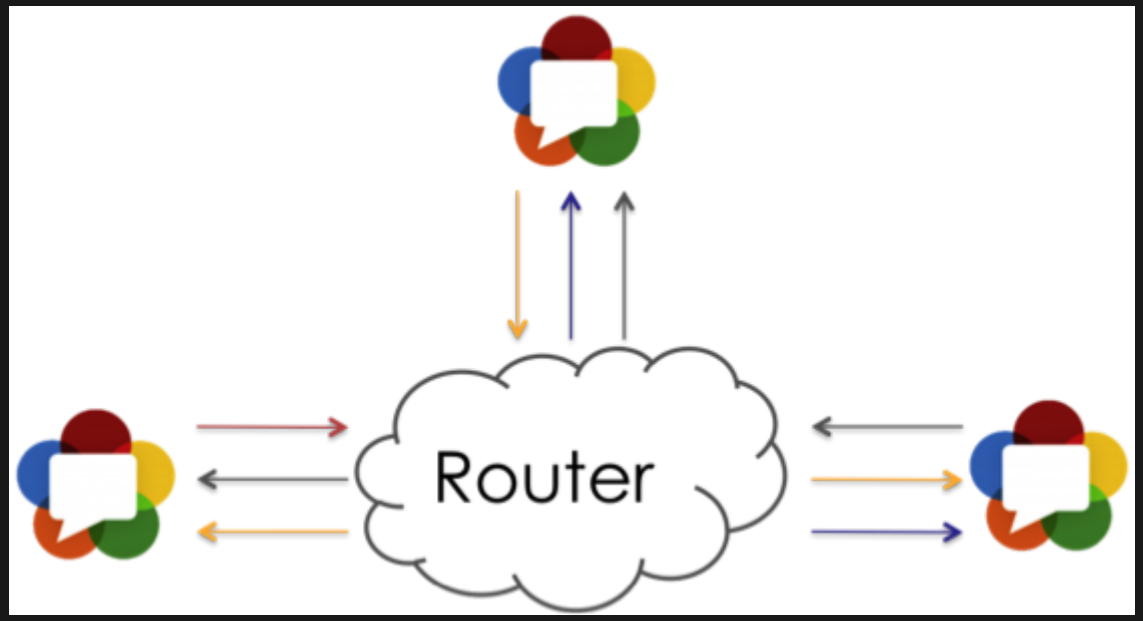


MCU



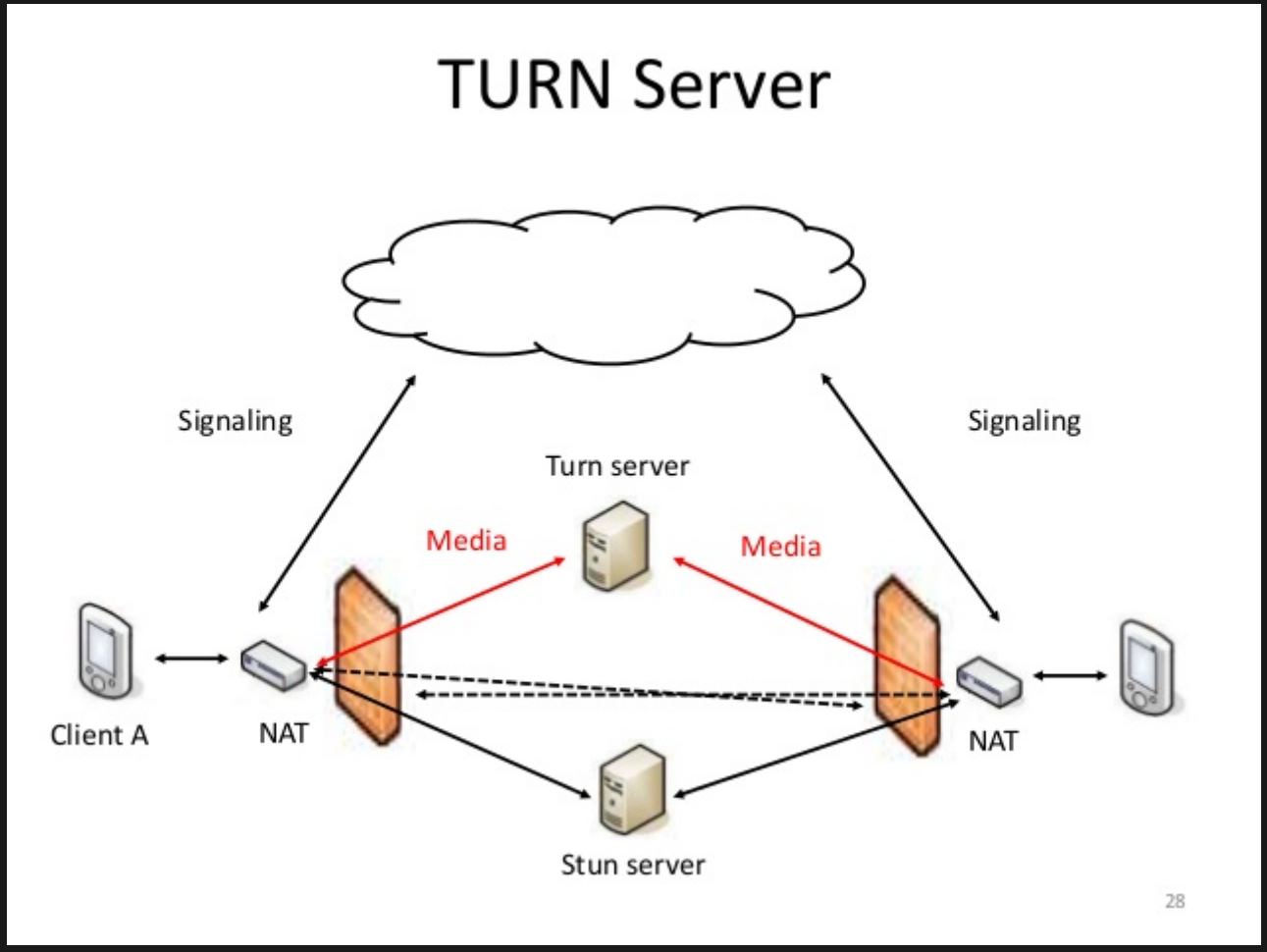
Multipoint Control Unit

SFU

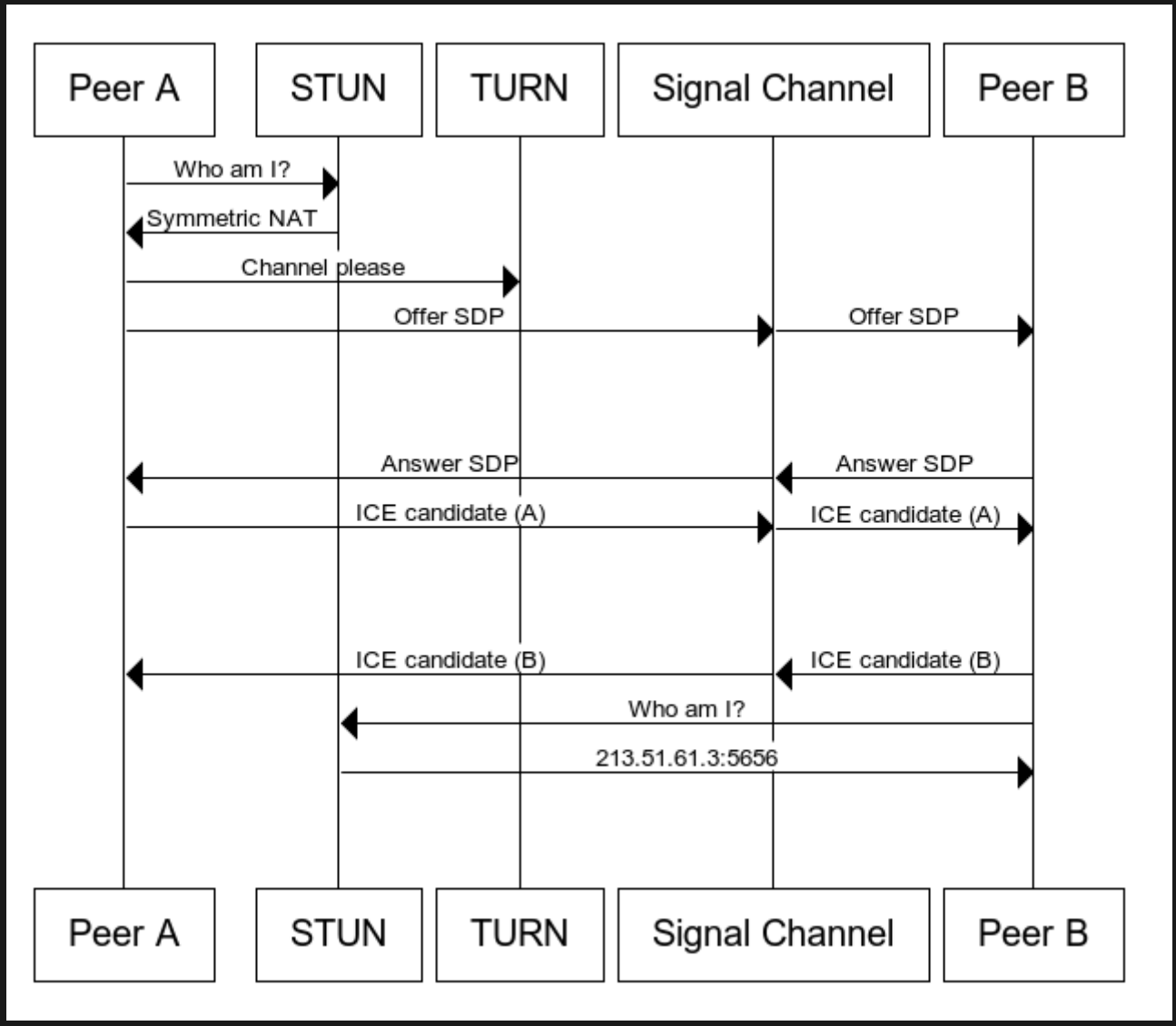


Selective Forwarding Unit

ATTRAVERSARE NAT



da help.nextcloud.com



da hacks.mozilla.org

DOMANDE?

GRAZIE