WebRTC broadcasting with WHIP

Lorenzo Miniero

@elminiero

FOSDEM 2022 Real Time Communications

5th February 2022, Brussels My couch
Who am I?

Lorenzo Miniero
- Ph.D @ UniNA
- Chairman @ Meetecho
- Main author of Janus

Contacts and info
- lorenzo@meetecho.com
- https://twitter.com/elminiero
- https://www.slideshare.net/LorenzoMiniero
- https://lminiero.bandcamp.com
MCU as a WebRTC topology
SFU as a WebRTC topology
A slightly variation on the theme
How “traditional” broadcasting typically works
What if we want more interactivity, though?
What if we want more interactivity, though?
What if we want more interactivity, though?
What if we want more interactivity, though?
Why not WebRTC instead?

- Traditional broadcasting efficient but higher latency
  - At best (live), delay will typically be in the range of a few seconds
  - Besides, different users may experience different delays (buffering)

- WebRTC natively conceived for very low latency, instead
  - Born for conversational audio/video/data
  - Can be (and often is) easily used for monodirectional streaming as well

- Strangely not really considered by the industry up until recently, though
  - Topic of my Ph.D years ago (“Streaming Of Large scale Events over Internet cLouds”)
  - Clearing the industry FUD: https://webrtcbydralex.com/index.php/2020/04/14/

- Tooling an important aspect to foster WebRTC adoption, here
  - e.g., a standard way to send media, and tools à la OBS, XSplit, or others
Why not WebRTC instead?

• Traditional broadcasting efficient but higher latency
  • At best (live), delay will typically be in the range of a few seconds
  • Besides, different users may experience different delays (buffering)

• WebRTC natively conceived for very low latency, instead
  • Born for conversational audio/video/data
  • Can be (and often is) easily used for monodirectional streaming as well

• Strangely not really considered by the industry up until recently, though
  • Topic of my Ph.D years ago (“Streaming Of Large scale Events over Internet cLouds”)  
  • Clearing the industry FUD: https://webrtcbydralex.com/index.php/2020/04/14/

• Tooling an important aspect to foster WebRTC adoption, here
  • e.g., a standard way to send media, and tools à la OBS, XSplit, or others
Why not WebRTC instead?

- Traditional broadcasting efficient but higher latency
  - At best (live), delay will typically be in the range of a few seconds
  - Besides, different users may experience different delays (buffering)
- WebRTC natively conceived for very low latency, instead
  - Born for conversational audio/video/data
  - Can be (and often is) easily used for monodirectional streaming as well
- Strangely not really considered by the industry up until recently, though
  - Topic of my Ph.D years ago ("Streaming Of Large scale Events over Internet cLouds")
  - Clearing the industry FUD: https://webrtcbydralex.com/index.php/2020/04/14/
- Tooling an important aspect to foster WebRTC adoption, here
  - e.g., a standard way to send media, and tools à la OBS, XSplit, or others
Why not WebRTC instead?

- Traditional broadcasting efficient but higher latency
  - At best (live), delay will typically be in the range of a few seconds
  - Besides, different users may experience different delays (buffering)
- WebRTC natively conceived for very low latency, instead
  - Born for conversational audio/video/data
  - Can be (and often is) easily used for monodirectional streaming as well
- Strangely not really considered by the industry up until recently, though
  - Topic of my Ph.D years ago (“Streaming Of Large scale Events over Internet cLouds”)
  - Clearing the industry FUD: https://webrtcbydralex.com/index.php/2020/04/14/
- Tooling an important aspect to foster WebRTC adoption, here
  - e.g., a standard way to send media, and tools à la OBS, XSplit, or others
WebRTC audio: Opus

- WebRTC mandates Opus, and it’s a good thing
  - High quality audio codec designed for the Internet
  - Very flexible in sampling rates, bitrates, FEC, etc.

- Different profiles for voice and music/other
  - Both encoding and decoding vary, in case
  - Can be mono and stereo, with dynamic sampling rates and bitrates

- Multiopus (5.1 and 7.1)\(^1\)
  - Each packet is basically OGG with multiple stereo Opus streams
  - Number of streams determines number of channels
  - Not documented, but used by Google for Stadia

WebRTC audio: Opus

- WebRTC mandates Opus, and it’s a good thing
  - High quality audio codec designed for the Internet
  - Very flexible in sampling rates, bitrates, FEC, etc.

- Different profiles for voice and music/other
  - Both encoding and decoding vary, in case
  - Can be mono and stereo, with dynamic sampling rates and bitrates

- Multiopus (5.1 and 7.1)\(^1\)
  - Each packet is basically OGG with multiple stereo Opus streams
  - Number of streams determines number of channels
  - Not documented, but used by Google for Stadia

WebRTC audio: Opus

- WebRTC mandates Opus, and it’s a good thing
  - High quality audio codec designed for the Internet
  - Very flexible in sampling rates, bitrates, FEC, etc.
- Different profiles for voice and music/other
  - Both encoding and decoding vary, in case
  - Can be mono and stereo, with dynamic sampling rates and bitrates
- Multiopus (5.1 and 7.1)\(^1\)
  - Each packet is basically OGG with multiple stereo Opus streams
  - Number of streams determines number of channels
  - Not documented, but used by Google for Stadia

WebRTC video: **Simulcasting** & SVC

[Link to SFU Simulcast](https://webrtcchacks.com/sfu-simulcast/)

---

**Diagram:**

- Two user icons are connected to a server, with green arrows indicating data flow.
- A blue line connects the server to another user icon, representing a different data flow or connection type.

---

**Note:**

- WebRTC is a technology that enables real-time communication between web applications. Simulcasting and SVC (Spatial Virtual Codec) are advanced features in WebRTC that enhance the user experience by allowing multiple streams to be sent and received simultaneously.
WebRTC video: Simulcasting & SVC

https://webrtcchacks.com/chrome-vp9-svc/
Two main challenges: ingestion...
Two main challenges: ... and distribution
(and yeah, maybe scaling too! 😊)
Making WebRTC ingestion easy: WISH / WHIP!

https://www.meetecho.com/blog/whip-janus-part-ii/
There would be no WHIP without Dr. Alex ❤️
A new Working Group in the IETF...

WebRTC Ingest Signaling over HTTPS (wish)

About Documents Meetings History Photos Email expansions List archive » Tools »

WG Name WebRTC Ingest Signaling over HTTPS
Acronym wish
Area Applications and Real-Time Area (art)
State Active
Charter charter-ietf-wish-01
Dependencies Document dependency graph (SVG)
Additional Github
Resources

Personnel Chairs Nils Ohlmeier Sean Turner
Area Director Murray Kucherawy

Mailing list Address wish@ietf.org
To subscribe https://www.ietf.org/mailman/listinfo/wish
Archive https://mailarchive.ietf.org/arch/browse/wish/

Jabber chat Room address xmppw wish@jabber.ietf.org?join
Logs https://jabber.ietf.org/logs/wish/

Charter for Working Group

The WISH working group is chartered to specify a simple, extensible, HTTPS-based signaling protocol to establish one-way WebRTC-based audiovisual sessions between broadcasting tools and real-time media broadcast networks.

https://datatracker.ietf.org/wg/wish/about/
... and a new draft for the WHIP specification!

WebRTC-HTTP ingestion protocol (WHIP)

Abstract

While WebRTC has been very successful in a wide range of scenarios, its adoption in the broadcasting/streaming industry is lagging behind. Currently there is no standard protocol (like SIP or RTSP) designed for ingesting media into a streaming service using WebRTC and so content providers still rely heavily on protocols like RTMP for it.

These protocols are much older than WebRTC and by default lack some important security and resilience features provided by WebRTC with minimal overhead and additional latency.

The media codecs used for ingestion in older protocols tend to be limited and not negotiable. WebRTC includes support for negotiation of codecs, potentially alleviating transcoding on the ingest node (which can introduce delay and degrade media quality). Server side transcoding that has traditionally been done to present multiple renditions in Adaptive Bit Rate Streaming (ABR) implementations can be replaced with simulcasting and SVC codecs that are well supported by WebRTC clients. In addition, WebRTC clients can adjust client-side encoding parameters based on RTCP feedback to maximize encoding quality.

Encryption is mandatory in WebRTC, therefore secure transport of media is implicit.

This document proposes a simple HTTP based protocol that will allow WebRTC based ingest of content into streaming services and/or CDNs.

https://www.ietf.org/archive/id/draft-ietf-wish-whip-01.html
WebRTC-HTTP ingestion protocol (WHIP)

- HTTP-based signalling to create *sendonly* PeerConnections
  - HTTP POST to send SDP offer, and get an SDP answer in the response
  - Teardown of sessions using HTTP DELETE
- Authentication and authorization via Bearer tokens
- Trickle and ICE restart via HTTP PATCH and SDP fragments
- Everything else is your usual WebRTC!
  - ICE, DTLS, etc.
WebRTC-HTTP ingestion protocol (WHIP)

- HTTP-based signalling to create `sendonly` PeerConnections
  - HTTP POST to send SDP offer, and get an SDP answer in the response
  - Teardown of sessions using HTTP DELETE
- Authentication and authorization via Bearer tokens
- Trickle and ICE restart via HTTP PATCH and SDP fragments
- Everything else is your usual WebRTC!
  - ICE, DTLS, etc.
WebRTC-HTTP ingestion protocol (WHIP)

- HTTP-based signalling to create *sendonly* PeerConnections
  - HTTP POST to send SDP offer, and get an SDP answer in the response
  - Teardown of sessions using HTTP DELETE
- Authentication and authorization via Bearer tokens
- Trickle and ICE restart via HTTP PATCH and SDP fragments
  - [https://www.rfc-editor.org/rfc/rfc8840.html](https://www.rfc-editor.org/rfc/rfc8840.html)
- Everything else is your usual WebRTC!
  - ICE, DTLS, etc.
WebRTC-HTTP ingestion protocol (WHIP)

- HTTP-based signalling to create `sendonly` PeerConnections
  - HTTP POST to send SDP offer, and get an SDP answer in the response
  - Teardown of sessions using HTTP DELETE

- Authentication and authorization via Bearer tokens

- Trickle and ICE restart via HTTP PATCH and SDP fragments

- Everything else is your usual WebRTC!
  - ICE, DTLS, etc.
A few sequence diagrams
A few sequence diagrams

HTTP POST (Bearer token + SDP Offer)

201 Accepted (Resource Location + SDP answer)

PeerConnection setup

RTP flow

perform an ICE restart

HTTP PATCH (new ICE credentials)

200 OK (new ICE credentials)

react to ICE restart
A few sequence diagrams

HTTP POST (Bearer token + SDP Offer)

201 Accepted (Resource Location + SDP answer)

PeerConnection setup

RTP flow

media flows

HTTP DELETE

get rid of PeerConnection

200 OK
A WHIP server based on Janus

• Janus is a popular WebRTC server, so good option for WHIP
  • It implements its own JSON-based API, though (Janus API)

• Simple (and transparent) solution: basic API translator in front of Janus
  • WHIP API maps quite simply to set of Janus API primitives
  • No need to change anything in the WebRTC stack

• Implemented simple prototype using node.js and Express
  • REST server that implements the WHIP API, and talks to Janus accordingly
  • Only takes care of ingest: distribution out of scope (e.g., via SOLEIL)

Simple WHIP Server
https://github.com/lminiero/simple-whip-server/
A WHIP server based on Janus

- Janus is a popular WebRTC server, so good option for WHIP
  - It implements its own JSON-based API, though (Janus API)
- Simple (and transparent) solution: basic API translator in front of Janus
  - WHIP API maps quite simply to set of Janus API primitives
  - No need to change anything in the WebRTC stack
- Implemented simple prototype using node.js and Express
  - REST server that implements the WHIP API, and talks to Janus accordingly
  - Only takes care of ingest: distribution out of scope (e.g., via SOLEIL)

Simple WHIP Server
https://github.com/lminiero/simple-whip-server/
A WHIP server based on Janus

- Janus is a popular WebRTC server, so good option for WHIP
  - It implements its own JSON-based API, though (Janus API)
- Simple (and transparent) solution: basic API translator in front of Janus
  - WHIP API maps quite simply to set of Janus API primitives
  - No need to change anything in the WebRTC stack
- Implemented simple prototype using node.js and Express
  - REST server that implements the WHIP API, and talks to Janus accordingly
  - Only takes care of ingest: distribution out of scope (e.g., via SOLEIL)

Simple WHIP Server
https://github.com/lminiero/simple-whip-server/
A WHIP server based on Janus

- Janus is a popular WebRTC server, so good option for WHIP
  - It implements its own JSON-based API, though (Janus API)
- Simple (and transparent) solution: basic API translator in front of Janus
  - WHIP API maps quite simply to set of Janus API primitives
  - No need to change anything in the WebRTC stack
- Implemented simple prototype using node.js and Express
  - REST server that implements the WHIP API, and talks to Janus accordingly
  - Only takes care of ingest: distribution out of scope (e.g., via SOLEIL)

Simple WHIP Server

https://github.com/lminiero/simple-whip-server/
Mapping WHIP interactions to the Janus API

- HTTP POST /endpoint (SDP Offer)
  - validate token
    - attach handle (VideoRoom)
      - success
      - joinandconfigure (SDP Offer)
        - ack
        - event (joined, SDP Answer)
  - add Location header to response for /resource
    - 201 Accepted (SDP answer)
      - address /resource from now on
Mapping WHIP interactions to the Janus API

HTTP PATCH /resource (trickle) -> find handle associated to /resource

204 No Content

ICE request -> ICE response

DTLS setup -> RTP flow

WebRTC Producer -> WHIP Server -> Janus
Mapping WHIP interactions to the Janus API

- WebRTC Producer
- WHIP Server
- Janus

HTTP PATCH /resource (new ufrag/pwd)

new credentials --> ICE restart, generate SDP

find handle associated to /resource

configure (SDP Offer, restart)

ack

event (configured, SDP Answer)

200 OK (new ufrag/pwd)

ICE request

ICE response
Mapping WHIP interactions to the Janus API

1. HTTP DELETE /resource
2. Cleanup endpoint
3. Find handle associated to /resource
4. Detach handle
5. Get rid of PeerConnection
6. DTLS alert
7. 200 OK
8. Success
Mapping WHIP interactions to the Janus API
Simple WHIP Server in action 😊

[iminiero@iminiero server]$ npm start

> janus-whip-server@0.0.1 start /home/iminiero/Work/code/services/whip/server
> DEBUG=whip:*,whip:debug,janus:*,janus:debug,janus:vdebug node src/server.js

[1. Janus]
Connecting to Janus: { address: 'ws://127.0.0.1:8188' }
  janus:info Connecting to ws://127.0.0.1:8188 +0ms
  janus:info Janus WebSocket Client Connected +7ms
  janus:info Janus session ID is 1252536092417283 +2ms
  whip:info Connected to Janus: ws://127.0.0.1:8100 +0ms

[2. WHIP REST API]
WHIP REST API listening on *:7080
WHIP server prototype started!
[ 'Janus OK', 'WHIP REST API OK' ]
  whip:info [ciao] Created new WHIP endpoint +32s
  whip:info [ciao] Publishing to WHIP endpoint +5s
  whip:info [ciao] Terminating WHIP session +12s
  whip:info [ciao] Publishing to WHIP endpoint +23s
  whip:info [ciao] PeerConnection detected as closed +8s
Basic UI to create/manage endpoints
Writing a WHIP client for testing

• Needs to support HTTP (WHIP API) and have a WebRTC stack
  • Browsers are the obvious choice, but what about a native solution?
  • Many broadcasters today use custom tools (e.g., OBS)

• Unfortunately OBS-WebRTC is not currently an option
  • Used legacy WHIP API, and currently only supports Millicast ingestion

• Chose GStreamer’s `webrtcbin`\(^2\) for the purpose
  • Used it already with success in other applications (e.g., JamRTC)
  • Modular and very powerful, so easy to feed with external sources

Simple WHIP Client

https://github.com/lminiero/simple-whip-client/

\(^2\)https://gstreamer.freedesktop.org/documentation/webrtc/
Writing a WHIP client for testing

- Needs to support HTTP (WHIP API) and have a WebRTC stack
  - Browsers are the obvious choice, but what about a native solution?
  - Many broadcasters today use custom tools (e.g., OBS)
- Unfortunately OBS-WebRTC is not currently an option
  - Used legacy WHIP API, and currently only supports Millicast ingestion
- Chose GStreamer’s `webrtcbin`\(^2\) for the purpose
  - Used it already with success in other applications (e.g., JamRTC)
  - Modular and very powerful, so easy to feed with external sources

**Simple WHIP Client**

[https://github.com/lminiero/simple-whip-client/](https://github.com/lminiero/simple-whip-client/)

\(^2\)[https://gstreamer.freedesktop.org/documentation/webRTC/]
Writing a WHIP client for testing

- Needs to support HTTP (WHIP API) and have a WebRTC stack
  - Browsers are the obvious choice, but what about a native solution?
  - Many broadcasters today use custom tools (e.g., OBS)
- Unfortunately OBS-WebRTC is not currently an option
  - Used legacy WHIP API, and currently only supports Millicast ingestion
- Chose GStreamer’s `webrtcbin`\(^2\) for the purpose
  - Used it already with success in other applications (e.g., JamRTC)
  - Modular and very powerful, so easy to feed with external sources

**Simple WHIP Client**

[https://github.com/lminiero/simple-whip-client/](https://github.com/lminiero/simple-whip-client/)

Writing a WHIP client for testing

• Needs to support HTTP (WHIP API) and have a WebRTC stack
  • Browsers are the obvious choice, but what about a native solution?
  • Many broadcasters today use custom tools (e.g., OBS)
• Unfortunately OBS-WebRTC is not currently an option
  • Used legacy WHIP API, and currently only supports Millicast ingestion
• Chose GStreamer’s `webrtcbin`\(^2\) for the purpose
  • Used it already with success in other applications (e.g., JamRTC)
  • Modular and very powerful, so easy to feed with external sources

Simple WHIP Client

https://github.com/lminiero/simple-whip-client/

\(^2\)https://gstreamer.freedesktop.org/documentation/webrtc/
Simple WHIP Client options

Usage:
whip-client [OPTION?] -- Simple WHIP client

Help Options:
-h, --help Show help options

Application Options:
-u, --url Address of the WHIP endpoint (required)
-t, --token Authentication Bearer token to use (optional)
-A, --audio GStreamer pipeline to use for audio (optional, required if audio-only)
-V, --video GStreamer pipeline to use for video (optional, required if video-only)
-n, --no-trickle Don’t trickle candidates, but put them in the SDP offer (default: false)
-f, --follow-link Use the Link headers returned by the WHIP server to automatically configure STUN/TURN servers to use (default: false)
-S, --stun-server STUN server to use, if any (stun://hostname:port)
-T, --turn-server TURN server to use, if any; can be called multiple times (turn(s)://username:password@host:port?transport=[udp,tcp])
-F, --force-turn In case TURN servers are provided, force using a relay (default: false)
-l, --log-level Logging level (0=disable logging, 7=maximum log level; default: 4)
Simple WHIP Client example

```
./whip-client -u http://localhost:7080/whip/endpoint/abc123 \
  -t verysecret \ 
  -A "audiotestsrc is-live=true wave=red-noise ! audioconvert ! 
  audioresample ! queue ! opusenc ! rtpopuspay pt=100 ! queue ! 
  application/x-rtp,media=audio,encoding-name=OPUS,payload=100" \
  -V "videotestsrc is-live=true pattern=ball ! videoconvert ! queue ! 
  vp8enc deadline=1 ! rtpvp8pay pt=96 ! queue ! 
  application/x-rtp,media=video,encoding-name=VP8,payload=96" \
  -S stun.l.google.com:19302
```
WHIP client and server + Janus
Simple WHIP Client in action 😊

```
[WHIP] Initializing the GStreamer pipeline:
webrtcsbin name=sendonly bundle-policy=3 videotests src is-live=true pattern=ball
! videoconvert ! queue ! vp8enc deadline=1 ! rtvp8pay pt=96 ssrc=2 ! queue ! application/x-rtp,media=video,encoding-name=VP8,payload=96 ! sendonly. audiotests src is-live=true wave=red-noise ! audioconvert ! audioresample ! queue ! opusenc ! rtppuspay pt=100 ssrc=1 ! queue ! application/x-rtp,media=audio,encoding-name=OPUS,payload=100 ! sendonly.
[WHIP] Starting the GStreamer pipeline
[WHIP] Creating offer
[WHIP] Offer created
[WHIP] Setting local description
[WHIP] Sending SDP offer (1167 bytes)
[WHIP] Received SDP answer (1385 bytes)
[WHIP] Setting remote description
[WHIP] ICE gathering started...
[WHIP] PeerConnection connecting...
[WHIP] ICE connecting...
[WHIP] ICE completed
[WHIP] DTLS connecting...
[WHIP] DTLS connected
[WHIP] PeerConnection connected
[WHIP] ICE gathering completed
```
Testing my WHIP client with Janus
Other WHIP implementations: servers

- Lorenzo Miniero – Simple WHIP Server (Janus)
  - https://github.com/lminiero/simple-whip-server

- Juliusz Chroboczek – Galene
  - https://github.com/jech/galene/tree/whip

- Sergio Garcia Murillo – Millicast integration
  - https://millicast.com/

- Cameron Elliott – Deadsfu
  - https://github.com/x186k/deadsfu
Other WHIP implementations: clients

- Lorenzo Miniero – Simple WHIP Client (GStreamer)
  - https://github.com/lminiero/simple-whip-client
- Sergio Garcia Murillo – whip-js (JavaScript)
  - https://github.com/medooze/whip-js/
- Gustavo Garcia – whip-go (Go)
  - https://github.com/ggarber/whip-go/
- Tim Panton – whipi (Java / Raspberry Pi)
  - https://github.com/pipe/whipi
- Alberto Gonzalez Trastoy – free-whip (Python)
  - https://github.com/agonza1/free-whip/
- Cameron Elliott – whip-whap-js (JavaScript)
  - https://github.com/x186k/whip-whap-js
WHIP Interoperability Tests

Lorenzo Miniero, on behalf of the group

IETF 112 Hackathon
November 5th, 2021

### WHIP interop tests @ IETF 112 Hackathon!

<table>
<thead>
<tr>
<th>Simple WHIP Client</th>
<th>Simple WHIP Server</th>
<th>Galene</th>
<th>Millicast</th>
<th>deadsfu</th>
</tr>
</thead>
<tbody>
<tr>
<td>whip-js</td>
<td>☀️</td>
<td>☀️</td>
<td>☀️</td>
<td>☀️</td>
</tr>
<tr>
<td>whip-go</td>
<td>☀️</td>
<td>☀️</td>
<td>☀️</td>
<td>☠️</td>
</tr>
<tr>
<td>whipi</td>
<td>☀️</td>
<td>☠️</td>
<td>☠️</td>
<td>☠️</td>
</tr>
<tr>
<td>free-whip</td>
<td>☠️</td>
<td>☠️</td>
<td>☠️</td>
<td>☠️</td>
</tr>
<tr>
<td>whip-whap-js</td>
<td>☠️</td>
<td>☠️</td>
<td>☠️</td>
<td>☠️</td>
</tr>
</tbody>
</table>
Integration WHIP in broadcasting workflows

- Old version of OBS-WebRTC did have WHIP support
  - Tested for my blog post from last year
  - Implemented legacy WHIP API, and used libwebrtc
- No popular streamer tool supports WHIP yet, though
  - WHIP will make it easy for the signalling part...
  - ... but they’ll still need a working WebRTC stack
- Why not start with a more “loose” integration then?
  - Keeping on using existing tools as they work today
  - Somehow get them to work with my GStreamer-based WHIP client
Integration WHIP in broadcasting workflows

- Old version of OBS-WebRTC did have WHIP support
  - Tested for my blog post from last year
  - Implemented legacy WHIP API, and used libwebrtc
- No popular streamer tool supports WHIP yet, though
  - WHIP will make it easy for the signalling part...
  - ... but they’ll still need a working WebRTC stack
- Why not start with a more “loose” integration then?
  - Keeping on using existing tools as they work today
  - Somehow get them to work with my GStreamer-based WHIP client
Integration WHIP in broadcasting workflows

- Old version of OBS-WebRTC did have WHIP support
  - Tested for my blog post from last year
  - Implemented legacy WHIP API, and used libwebrtc
- No popular streamer tool supports WHIP yet, though
  - WHIP will make it easy for the signalling part...
  - ... but they’ll still need a working WebRTC stack
- Why not start with a more “loose” integration then?
  - Keeping on using existing tools as they work today
  - Somehow get them to work with my GStreamer-based WHIP client
Enter NDI!

https://www.meetecho.com/blog/webrtc-ndi/
https://www.meetecho.com/blog/webrtc-ndi-part-2/
What is NDI?

- Network Device Interface (NDI)
  - Royalty-free software standard developed by NewTek
  - [https://www.ndi.tv/](https://www.ndi.tv/)

- Live exchange of multimedia streams within the same LAN
  - Multichannel and uncompressed media streams (high quality)
  - mDNS used for service discovery

- Easy to use (and integrate) native SDK
  - Available on Windows, Linux, MacOS, Android, etc.
  - VLC team working on an alternative implementation

- Widely used in the broadcasters industry
  - Natively supported by many devices and streamer tools
What is NDI?

- Network Device Interface (NDI)
  - Royalty-free software standard developed by NewTek
  - [https://www.ndi.tv/](https://www.ndi.tv/)

- Live exchange of multimedia streams within the same LAN
  - Multichannel and uncompressed media streams (high quality)
  - mDNS used for service discovery

- Easy to use (and integrate) native SDK
  - Available on Windows, Linux, MacOS, Android, etc.
  - VLC team working on an alternative implementation

- Widely used in the broadcasters industry
  - Natively supported by many devices and streamer tools
What is NDI?

- Network Device Interface (NDI)
  - Royalty-free software standard developed by NewTek
  - https://www.ndi.tv/
- Live exchange of multimedia streams within the same LAN
  - Multichannel and uncompressed media streams (high quality)
  - mDNS used for service discovery
- Easy to use (and integrate) native SDK
  - Available on Windows, Linux, MacOS, Android, etc.
  - VLC team working on an alternative implementation
- Widely used in the broadcasters industry
  - Natively supported by many devices and streamer tools
What is NDI?

- Network Device Interface (NDI)
  - Royalty-free software standard developed by NewTek
  - https://www.ndi.tv/
- Live exchange of multimedia streams within the same LAN
  - Multichannel and uncompressed media streams (high quality)
  - mDNS used for service discovery
- Easy to use (and integrate) native SDK
  - Available on Windows, Linux, MacOS, Android, etc.
  - VLC team working on an alternative implementation
- Widely used in the broadcasters industry
  - Natively supported by many devices and streamer tools
We’ve talked about WebRTC-to-NDI before...
What about NDI-to-WebRTC, though?

- There's a cool NDI plugin for GStreamer
  - Makes it easy to use NDI sources in GStreamer pipelines
  - [https://github.com/teltek/gst-plugin-ndi](https://github.com/teltek/gst-plugin-ndi)

- Hey, our WHIP client is based on GStreamer too!
  - Audio and video pipelines are customizable (command line)
  - NDI plugin as source for the media $\rightarrow$ encoders/WebRTC will do the rest

- Of course, we need something that generates NDI
  - OBS has an NDI plugin, for NDI input and output
  - [https://github.com/Palakis/obs-ndi](https://github.com/Palakis/obs-ndi)
What about NDI-to-WebRTC, though?

- There’s a cool NDI plugin for GStreamer
  - Makes it easy to use NDI sources in GStreamer pipelines
  - [https://github.com/teltek/gst-plugin-ndi](https://github.com/teltek/gst-plugin-ndi)
- Hey, our WHIP client is based on GStreamer too!
  - Audio and video pipelines are customizable (command line)
  - NDI plugin as source for the media → encoders/WebRTC will do the rest
- Of course, we need something that generates NDI
  - OBS has an NDI plugin, for NDI input and output
  - [https://github.com/Palakis/obs-ndi](https://github.com/Palakis/obs-ndi)
What about NDI-to-WebRTC, though?

- There’s a cool NDI plugin for GStreamer
  - Makes it easy to use NDI sources in GStreamer pipelines
  - [https://github.com/teltek/gst-plugin-ndi](https://github.com/teltek/gst-plugin-ndi)

- Hey, our WHIP client is based on GStreamer too!
  - Audio and video pipelines are customizable (command line)
  - NDI plugin as source for the media → encoders/WebRTC will do the rest

- Of course, we need something that generates NDI
  - OBS has an NDI plugin, for NDI input *and* output
  - [https://github.com/Palakis/obs-ndi](https://github.com/Palakis/obs-ndi)
1. Configure NDI output in OBS
2. Create your scenes in OBS
3. Setup the WHIP client to capture NDI

```bash
[WHIP] Initializing the GStreamer pipeline:
webrtcbin name=sendonly bundle-policy=3 ndisrc ndi-name=\"lminiero (OBS)\" ! ndisrcdemux name=n.video ! videoconvert ! vp8enc deadline=1 target-bitrate=500000 ! rtpvppay pt=96 ssrc=2 ! queue ! application/x-rtp,media=video,encoding-name=VP8,payload=96 ! sendonly.
n.audio ! audioconvert ! audioresources ! audiosink
[WHIP] Starting the GStreamer pipeline
[WHIP] Creating offer
[WHIP] Offer created
[WHIP] Setting local description
[WHIP] Sending SDP offer (1163 bytes)
[WHIP] Received SDP answer (1497 bytes)
[WHIP] Setting remote description
[WHIP] ICE gathering started...
[WHIP] PeerConnection connecting...
[WHIP] ICE connecting...
[WHIP] ICE completed
[WHIP] DTLS connecting...
[WHIP] DTLS connected
[WHIP] PeerConnection connected
[WHIP] ICE gathering completed
```
Ready for our demo!
Ready for our demo!
Ready for our demo!
More details in a recent CommCon talk

Next step: broadcasting the stream

- WHIP server + Janus get you in a VideoRoom, and it’s a good starting point
  - That’s the whole point of WebRTC ingest!
  - Already “consumable” via VideoRoom itself (SFU)
- Janus VideoRoom plugin not really optimized for broadcasting, though
  - Conceived for videoconferencing use cases
  - Will not work well if you have to feed, e.g., 100’s or 1000’s
- Janus Streaming plugin a much better choice
  - Natively optimized for doing one-to-many
  - Can receive media from VideoRoom (and so WHIP) via “RTP forwarders”
  - Even better, multiple Janus instances can work together
Next step: broadcasting the stream

- WHIP server + Janus get you in a VideoRoom, and it’s a good starting point
  - That’s the whole point of WebRTC ingest!
  - Already “consumable” via VideoRoom itself (SFU)
- Janus VideoRoom plugin not really optimized for broadcasting, though
  - Conceived for videoconferencing use cases
  - Will not work well if you have to feed, e.g., 100’s or 1000’s
- Janus Streaming plugin a much better choice
  - Natively optimized for doing one-to-many
  - Can receive media from VideoRoom (and so WHIP) via “RTP forwarders”
  - Even better, multiple Janus instances can work together
Next step: broadcasting the stream

- WHIP server + Janus get you in a VideoRoom, and it’s a good starting point
  - That’s the whole point of WebRTC ingest!
  - Already “consumable” via VideoRoom itself (SFU)
- Janus VideoRoom plugin not really optimized for broadcasting, though
  - Conceived for videoconferencing use cases
  - Will not work well if you have to feed, e.g., 100’s or 1000’s
- Janus Streaming plugin a much better choice
  - Natively optimized for doing one-to-many
  - Can receive media from VideoRoom (and so WHIP) via “RTP forwarders”
  - Even better, multiple Janus instances can work together
Università degli Studi di Napoli Federico II
Facoltà di Ingegneria

Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XXVII Ciclo
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell’Informazione

SOLEIL: Streaming Of Large scale Events over Internet cLouds
Lorenzo Miniero
Ph.D. Thesis
“Streaming Of Large scale Events over Internet cLouds” (Ph.D Thesis)

- In a nutshell, tree-based distribution of WebRTC feeds
- Ingest and edges are WebRTC (Janus), everything in the middle just RTP

- Working with RTP in intermediate layers has many advantages
  - No WebRTC overhead, and easier to route/manipulate by non-WebRTC tools
  - You can even take advantage of multicast, here

- Just needs RTP forwarding to start everything
  - PR available in WHIP server to do RTP forwarding (merged)
  - https://github.com/lminiero/simple-whip-server/pull/2
Using SOLEIL for the purpose

- “Streaming Of Large scale Events over Internet cLouds” (Ph.D Thesis)
  - In a nutshell, tree-based distribution of WebRTC feeds
  - Ingest and edges are WebRTC (Janus), everything in the middle just RTP

- Working with RTP in intermediate layers has many advantages
  - No WebRTC overhead, and easier to route/manipulate by non-WebRTC tools
  - You can even take advantage of multicast, here

- Just needs RTP forwarding to start everything
  - PR available in WHIP server to do RTP forwarding (merged)
  - https://github.com/lminiero/simple-whip-server/pull/2
• “Streaming Of Large scale Events over Internet cLouds” (Ph.D Thesis)
  • In a nutshell, tree-based distribution of WebRTC feeds
  • Ingest and edges are WebRTC (Janus), everything in the middle just RTP

• Working with RTP in intermediate layers has many advantages
  • No WebRTC overhead, and easier to route/manipulate by non-WebRTC tools
  • You can even take advantage of multicast, here

• Just needs RTP forwarding to start everything
  • PR available in WHIP server to do RTP forwarding (merged)
  • https://github.com/lminiero/simple-whip-server/pull/2
To learn more about RTP forwarders...

Janus as a WebRTC “enabler”
Having fun with RTP and external applications

Lorenzo Miniero
@elminiero

FOSDEM 2020 Real Time devroom
2nd February 2020, Brussels

https://archive.fosdem.org/2020/schedule/event/janus/
To learn more about RTP forwarders...

https://archive.fosdem.org/2020/schedule/event/janus/
To learn more about RTP forwarders...

https://archive.fosdem.org/2020/schedule/event/janus/
Distributing WHIP Janus streams via SOLEIL
Distributing WHIP Janus streams via SOLEIL
Distributing WHIP Janus streams via SOLEIL
Distributing WHIP Janus streams via SOLEIL
Distributing WHIP Janus streams via SOLEIL
Leveraging multicast internally for RTP

- **WebRTC**
  - **JANUS**
    - **VideoRoom**
    - **multicast**
    - **JANUS #2**
      - **Streaming**
    - **JANUS #3**
      - **Streaming**
Leveraging multicast internally for RTP