VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Telecommunications

Course: Voice over IP I (VoIP I)
Semestral Project (40 points)

OPENSOURCE VOIP VIDEOCONFERENCE

Implementation of a Multi-Party Video Conference Solution Using Open-Source VoIP SIP & WebRTC Tools

Author	Julian Lara (lar0053)
Lecturer	Ing. Filip Řezáč, Ph.D.
Date	November 2026
Project portal	https://web-conference-lar0053.duckdns.org

Contents

1. Introduction **3**

1.1 Objectives 3

2. State of the art **3**

2.1 Mesh, MCU and SFU 3

2.2 WebRTC and the SIP gateway 4

2.3 NAT traversal with ICE 5

2.4 Security of SIP and RTP 5

3. Technology and configuration **6**

3.1 Asterisk architecture 6

3.2 Deployment topology 7

3.3 Configuration listings 7

3.4 The browser client 12

4. Demonstration and analysis **12**

4.1 How the test was run 12

4.2 Registration and call set-up 12

4.3 Server state during the conference 14

4.4 Bandwidth analysis 18

4.5 Other rooms and clients 19

5. Own opinion and contribution **20**

5.1 ConfBridge SFU compared with dedicated SFUs 20

5.2 Problems found and how they were solved 20

5.3 Contribution 21

5.4 Security notes 21

6. Conclusion **21**

7. References **21**

1. Introduction

Voice over IP started as a replacement for the circuit-switched telephone: one codec, one RTP stream in each direction, and a signalling protocol (SIP, RFC 3261) that set the call up and tore it down. Two things changed that picture. Cameras and screens became cheap enough that video is the default expectation, and browsers gained a real-time media stack (WebRTC), so the client no longer has to be a hardware phone or a native softphone. A modern conference has to work from a laptop tab, a phone app and an old SIP desk phone at the same time, and it has to work with more than two people in the call.

That last requirement is where the simple model breaks. A point-to-point call has one signalling dialog and one pair of media streams. A conference with N participants has N dialogs on the server side, N×(N−1) potential media flows, and a bandwidth bill that depends heavily on where the streams are mixed or forwarded. On top of that come the practical problems this project had to solve on a real public server:

- signalling state: each participant must be registered, authenticated and identified, and the conference must survive participants joining and leaving while others are talking;
- NAT traversal: the server sits on a cloud VM with a private address (10.0.0.88) and a public one (82.70.77.39), and the clients sit behind home or university NAT;
- media encryption: browsers refuse to send media unencrypted, so the server must speak DTLS-SRTP while old SIP phones may still use plain RTP or SDES;
- bandwidth: every extra participant adds a video stream to every other participant's downlink;
- browser integration: SIP has to travel over WebSocket, and the page has to handle several remote video tracks that appear and disappear after renegotiation.

This report documents a working multi-party video conference built only from open-source components: Asterisk 20 (chan_pjsip and the ConfBridge application in SFU mode), nginx with a Let's Encrypt certificate, SIP.js in the browser, and standard softphones (Linphone, MicroSIP, sipML5) as alternative clients. The system is live at `https://web-conference-lar0053.duckdns.org`.

1.1 Objectives

The goal set by the assignment (topic 6, *Opensource VoIP videoconference*) is a SIP video call between three or more participants using open-source tools. The concrete objectives were: install and configure the server; register at least five accounts; let three or more participants see and hear each other; give the audience a web client that works without installing anything; measure what the conference costs in bandwidth; and report honestly what did not work.

2. State of the art

2.1 Mesh, MCU and SFU

There are three ways to connect N participants. In a **full mesh** every client sends its stream directly to every other client. An **MCU** (multipoint control unit) decodes all incoming streams, mixes them into one composite picture and one audio mix, encodes the result and sends one stream back to each client. An **SFU** (selective forwarding unit) does not decode anything: it receives one stream from each client and forwards a copy to each of the others, choosing which ones to send.

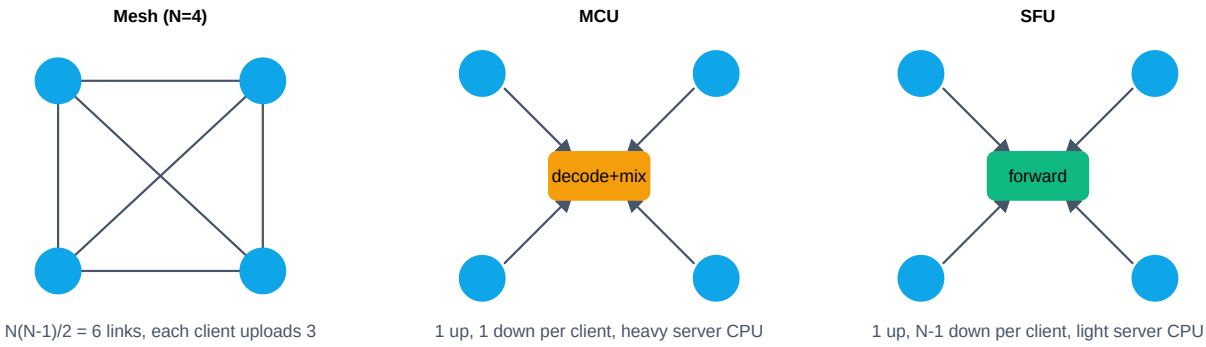


Figure 1. Three conference topologies.

Let *b* be the bitrate of one encoded stream (audio plus video, including packet headers). The per-client and per-server load follows directly from the topology:

Model	Client uplink	Client downlink	Server in	Server out	Server CPU	Latency
Mesh	(N−1)·b	(N−1)·b	0	0	none	lowest, one hop
MCU	b	b	N·b	N·b	decode N + encode N, high	highest, transcoding delay
SFU	b	(N−1)·b	N·b	N(N−1)·b	low, copy and rewrite headers	low, one extra hop

The mesh is attractive for N=3 and hopeless beyond N=5, because the uplink of a home connection and the encoder load on the client both grow linearly with N. The MCU gives the lightest client but makes the server the bottleneck: encoding five 720p streams in real time needs several CPU cores, and the delay of decoding, mixing and re-encoding adds tens of milliseconds. The SFU keeps the server cheap, at the price of a downlink that grows with N. Modern SFUs add simulcast or scalable video coding so that a client with a weak link can ask for a lower-resolution copy.

2.2 WebRTC and the SIP gateway

WebRTC gives a web page three things: `getUserMedia()` for camera and microphone, `RTCPeerConnection` for the media transport, and `RTCDataChannel` for arbitrary data. It deliberately does not define signalling. The page exchanges session descriptions with the other side using JSEP (the JavaScript Session Establishment Protocol): `createOffer()`, `setLocalDescription()`, then `setRemoteDescription()` with the answer. What carries those descriptions is the application's choice. Here it is SIP over a WebSocket (RFC 7118): each SIP message is one WebSocket text frame, the sub-protocol is `sip`, and the Via header uses transport `WSS`. SIP.js implements that on the page; on the server, Asterisk's `res_pjsip_transport_websocket` terminates it.

Modern browsers describe their media with **Unified Plan** SDP (RFC 8866 syntax). Each track gets its own `m=` line with a `mid`, and all of them are multiplexed on one transport with `a=group:BUNDLE` and `a=rtcp-mux`. This matters for conferencing because a new participant is announced by adding an `m=video` section in a re-INVITE rather than by changing an existing stream.

Aspect	Classic VoIP	WebRTC
Audio codecs	G.711 (PCMU/PCMA), G.729, GSM, iLBC	Opus mandatory, G.711 mandatory fallback; iSAC and iLBC in some browsers
Video codecs	H.263, H.264	VP8 and H.264 mandatory, VP9 and AV1 optional
Media security	optional (SDES, ZRTP, plain RTP)	mandatory SRTP keyed by DTLS
Signalling	SIP over UDP, TCP or TLS	not specified; here SIP over WSS
NAT traversal	symmetric RTP, STUN, SBC	ICE with STUN and TURN, mandatory
Multiplexing	separate ports for RTP and RTCP, audio and video	BUNDLE and rtcp-mux, one port

2.3 NAT traversal with ICE

ICE (RFC 8445) lets two endpoints find a working path through NAT. Each side gathers *candidates*: **host** candidates are its own interface addresses, **server-reflexive** (srflx) candidates are the public address a STUN server reports, and **relay** candidates are addresses allocated on a TURN server that forwards the media when no direct path exists. Both sides exchange candidates in the SDP, test pairs with STUN binding requests, and nominate the pair with the best priority that works. In this deployment Asterisk advertises a host candidate (10.0.0.88) and a srflx candidate (82.70.77.39) obtained from `stun.l.google.com`; the browsers do the same. No TURN server is deployed, so a client behind a symmetric NAT or a firewall that blocks UDP would not connect (see section 5).

2.4 Security of SIP and RTP

RFC 3261 requires that a proxy and a registrar be able to authenticate a user. The mechanism is HTTP Digest. The server answers an unauthenticated request with `401 Unauthorized` and a challenge containing a realm and a nonce. The client computes

> A1 = username:realm:password
> A2 = method:digest-uri
> response = MD5(MD5(A1) : nonce : MD5(A2))

and repeats the request with an `Authorization` header. With `qop=auth` the hash also includes a client nonce and a counter, `MD5(MD5(A1):nonce:nc:cnonce:qop:MD5(A2))`, which is what Asterisk asked for in the traces of section 4. The password never travels, but the exchange is only as private as the channel: signalling should run over TLS (the `sips:` scheme, or WSS for WebSocket), otherwise an eavesdropper can read the whole dialog and mount an offline guessing attack on the hash. Whole-message protection is also possible with S/MIME bodies, and the network layer can be protected with IPsec (AH for integrity, ESP for integrity and confidentiality), which suits site-to-site trunks more than end-user devices.

For the media, SRTP (RFC 3711) wraps each RTP packet: the header stays readable, the payload is encrypted, and an authentication tag is appended.

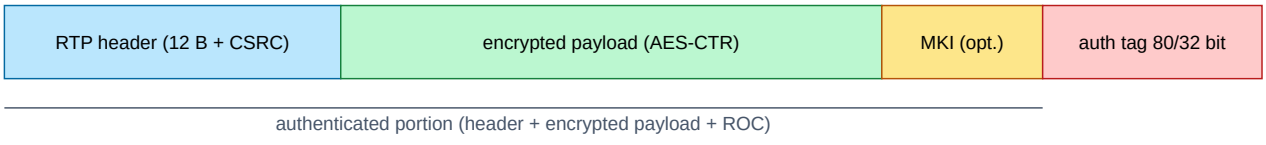


Figure 2. SRTP packet.

The default cipher is AES in counter mode. The keystream is produced from an initialisation vector built from the session salt k_s, the stream's SSRC and the packet index i (the 32-bit rollover counter concatenated with the 16-bit sequence number):

$$\mathrm{IV} = (k_s \cdot 2^{16}) \oplus (\mathrm{SSRC} \cdot 2^{64}) \oplus (i \cdot 2^{16})$$

Session keys (encryption, authentication, salt) are derived from a master key and master salt with a key-derivation function based on AES, so the master key can stay in use for a long time. Integrity comes from HMAC-SHA1 truncated to 80 bits (the default, seen in the inspector as `AES_CM_128_HMAC_SHA1_80`) or 32 bits for narrow links. Three schemes deliver the master key:

Scheme	RFC	How the key is agreed	Weak point
SDES	4568	Key sent in an `a=crypto` SDP line	Key visible to every signalling hop unless SIP runs over TLS
ZRTP	6189	Diffie-Hellman inside the media path; users compare a short authentication string (SAS); a retained secret (RS) is cached for later calls	Needs support in both endpoints; SAS check relies on the users
DTLS-SRTP	5764	DTLS handshake on the media port; the SDP carries only the certificate fingerprint (`a=fingerprint`)	Fingerprint must travel over an authenticated signalling channel

WebRTC uses DTLS-SRTP only. The fingerprint in the SDP binds the DTLS certificate to the signalling session, which is why the WSS transport matters: with plain `ws://` a man in the middle could swap fingerprints.

3. Technology and configuration

3.1 Asterisk architecture

Asterisk is a modular telephony engine. A small core manages channels, a scheduler and the module loader. Everything else is a module: channel drivers (`chan_pjsip`), codec translators (`codec_opus`), applications (`app_confbridge`, `Echo`, `Dial`) and resources (`res_http_websocket`, `res_rtp_asterisk`). A call arrives on a channel, the dialplan (`extensions.conf`) decides what to run for the dialled extension, and the *bridging* framework joins channels together, either two-party (native or through the core) or multi-party through ConfBridge. Programs can control the system from outside through AGI (per-call scripts), AMI (event and command socket) and ARI (REST plus WebSocket for building own applications).

chan_pjsip, built on the PJSIP library, replaced the old chan_sip. It separates the configuration into objects: *transport* (how to listen), *endpoint* (how a device behaves: codecs, context, encryption), *auth* (credentials), *aor* (where the device can be reached and how many registrations are kept), and *identify* (how an incoming request is matched to an endpoint). The old module had to be disabled with `noload => chan_sip.so`; left loaded, it claimed the `sip` WebSocket sub-protocol and every WebRTC REGISTER failed with 403 (section 5).

FreeSWITCH is the usual alternative. It is built around a state-machine core with a mod_verto and mod_sofia stack, a very capable conference module (`mod_conference`) with a mixing MCU, and an event-socket interface. Asterisk was chosen here because the course material is based on it, because ConfBridge gained a true SFU mode in version 16, and because the Ubuntu package is available for ARM64 without compiling.

3.2 Deployment topology

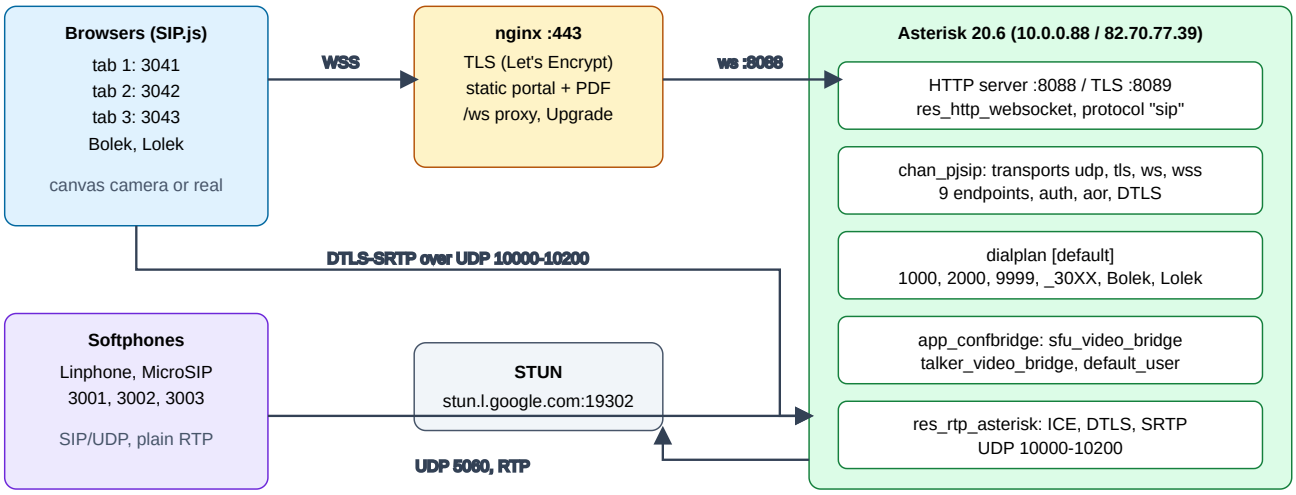


Figure 3. Deployment topology. Signalling reaches Asterisk through nginx on port 443; media goes straight to the RTP range.

The server is an Oracle Cloud ARM64 VM (Ubuntu 24.04, 4 CPU, 24 GB). nginx already served other sites on the same machine and was kept untouched except for a new virtual host and a `/ws` location. The host firewall (iptables) accepts the ports needed by Asterisk before the final REJECT rule and the rules are persisted with netfilter-persistent. The cloud provider's own security list is a separate layer that cannot be changed from the VM.

3.3 Configuration listings

http.conf enables Asterisk's built-in HTTP server. Port 8088 is plain HTTP and only reachable from nginx on the same host; 8089 is HTTPS with the certificate copied from Let's Encrypt. Browsers normally connect to port 443 through nginx, which avoids opening 8089.

```
[general]
servername=Asterisk-VoIP1
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.pem
tlsprivatekey=/etc/asterisk/keys/asterisk.key
```

rtp.conf fixes the media port range to 201 ports, enables ICE and gives Asterisk a STUN server so it can learn its public address.

```
[general]
rtpstart=10000
rtpend=10200
icesupport=yes
stunaddr=stun.l.google.com:19302
strictrtp=yes
rtpchecksums=no
```

pjsip.conf holds the transports, two endpoint templates and the accounts. The template `webrtc-endpoint` uses `webrtc=yes`, which switches on AVPF feedback, rtcp-mux, ICE and DTLS in one go, and permits both Opus and G.711 plus VP8 and H.264. The two `external_*` addresses make Asterisk write its public IP in SDP and Contact headers,

while `local_net` tells it which destinations are inside the VM's network and need no rewriting. Each account is an auth, an aor and an endpoint that share one name; the listing shows 3041 as an example (the other five WebRTC accounts and the three classic SIP accounts follow the same pattern).

```
; pjsip.conf - VoIP I videoconference (lar0053)
; Public IP 82.70.77.39, private IP 10.0.0.88 (Oracle Cloud NAT)

[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0:5060
external_media_address=82.70.77.39
external_signaling_address=82.70.77.39
local_net=10.0.0.0/24
local_net=127.0.0.0/8

[transport-tls]
type=transport
protocol=tls
bind=0.0.0.0:5061
cert_file=/etc/asterisk/keys/asterisk.pem
priv_key_file=/etc/asterisk/keys/asterisk.key
method=tlsv1_2
external_media_address=82.70.77.39
external_signaling_address=82.70.77.39
local_net=10.0.0.0/24
local_net=127.0.0.0/8

[transport-ws]
type=transport
protocol=ws
bind=0.0.0.0
external_media_address=82.70.77.39
external_signaling_address=82.70.77.39
local_net=10.0.0.0/24
local_net=127.0.0.0/8

[transport-wss]
type=transport
protocol=wss
bind=0.0.0.0
external_media_address=82.70.77.39
external_signaling_address=82.70.77.39
local_net=10.0.0.0/24
local_net=127.0.0.0/8

; ---------- WebRTC endpoint template ----------
[webrtc-endpoint](!)
type=endpoint
context=default
webrtc=yes
dtls_cert_file=/etc/asterisk/keys/asterisk.pem
dtls_private_key=/etc/asterisk/keys/asterisk.key
dtls_verify=fingerprint
dtls_setup=actpass
disallow=all
allow=opus,ulaw,alaw,vp8,h264
max_audio_streams=16
max_video_streams=16
direct_media=no
rtp_symmetric=yes
force_rport=yes
```

```
rewrite_contact=yes

; ---------- Classic SIP template (Linphone, MicroSIP) ----------
[sip-endpoint](!)
type=endpoint
context=default
transport=transport-udp
disallow=all
allow=ulaw,alaw,vp8,h264
media_encryption=no
direct_media=no
rtp_symmetric=yes
force_rport=yes
rewrite_contact=yes
max_video_streams=4

; ... same pattern for 3042, 3043, 3044, Bolek, Lolek ...

[3041]
type=auth
auth_type=userpass
username=3041
password=3041ab

[3041]
type=aor
max_contacts=5
remove_existing=yes

[3041](webrtc-endpoint)
auth=3041
aors=3041
callerid="3041" <3041>

; ... classic SIP accounts 3001-3003 use [sip-endpoint](!) ...
```

confbridge.conf defines two bridge profiles and one user profile. `video_mode=sfu` forwards each participant's video to all the others; `follow_talker` shows only the current speaker to everybody. `dsp_drop_silence` removes silent audio from the mix and the two event options feed the talker detection.

```
[general]

[sfu_video_bridge]
type=bridge
video_mode=sfu
max_members=10
internal_sample_rate=auto
mixing_interval=20

[talker_video_bridge]
type=bridge
video_mode=follow_talker
max_members=10
internal_sample_rate=auto
mixing_interval=20

[default_user]
type=user
dsp_drop_silence=yes
talk_detection_events=yes
echo_events=yes
```

extensions.conf maps numbers to actions. 1000 and 2000 answer and enter the conference, 9999 is an echo test that reflects the caller's own audio and video, and the last three patterns dial a registered endpoint directly.

```
[general]
static=yes
writeprotect=no

[default]
; SFU video conference
exten => 1000,1,Answer()
 same => n,ConfBridge(1000,sfu_video_bridge,default_user)
 same => n,Hangup()

; Follow-the-talker video conference
exten => 2000,1,Answer()
 same => n,ConfBridge(2000,talker_video_bridge,default_user)
 same => n,Hangup()

; Echo test
exten => 9999,1,Answer()
 same => n,Echo()
 same => n,Hangup()

; Extensions 3000-3099 and named users
exten => _30XX,1,Dial(PJSIP/${EXTEN},30)
 same => n,Hangup()
exten => Bolek,1,Dial(PJSIP/${EXTEN},30)
 same => n,Hangup()
exten => Lolek,1,Dial(PJSIP/${EXTEN},30)
 same => n,Hangup()
```

nginx proxies the WebSocket on the standard HTTPS port. The `Upgrade` and `Connection` headers must be forwarded explicitly, otherwise nginx answers the handshake itself, and the long read timeout stops it from closing an idle but registered connection.

```
location /ws {
    proxy_pass http://127.0.0.1:8088/ws;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_read_timeout 3600s;
}
```

3.4 The browser client

The page `/studio.html` uses SIP.js 0.21.2, bundled into a single local file with esbuild so that no CDN is needed, and Tailwind CSS from a local copy. A `UserAgent` opens the WSS connection and a `Registerer` sends the REGISTER; SIP.js answers the 401 challenge with the Digest computation of section 2.4. Calls are made with an `Inviter` whose session description handler owns the `RTCPeerConnection`. Three details are specific to this project:

- Every time Asterisk adds a participant it sends a re-INVITE with a new `m=video` section. SIP.js accepts it, the peer connection fires a `track` event, and the page puts each remote video track in its own tile and the audio track in a hidden audio element. When a track ends, the tile is removed.
- The simulated camera draws an animated pattern, the clock, a bouncing marker and the extension number on a 640×360 canvas at 30 fps and captures it with `captureStream(30)`. A Web Audio oscillator provides a short tone every three seconds, with a different pitch per extension. The page replaces `getUserMedia` with a function that returns this stream, so SIP.js needs no changes. A laptop can then host several participants in separate tabs, which a physical camera cannot do because only one process can open it.
- Muting, camera off and screen sharing act on the `RTCRtpSender` (`track.enabled` and `replaceTrack`), so they need no renegotiation. The inspector polls `getStats()` once a second for the codecs, frame size, ICE state, DTLS state and bitrate.

4. Demonstration and analysis

4.1 How the test was run

Five browser sessions (3041, 3042, 3043, Bolek and Lolek) were opened from one laptop on the university network (public address 158.196.230.193) with Chrome 148, each loading `studio.html` with the simulated camera. Every session registered over WSS and dialled 1000. They joined at four-second intervals. The CLI output below was captured on the server while all five were in the room. To repeat it by hand, open the studio in three tabs, click 3041, 3042 and 3043 in the respective tabs, press *Register* and then *Room 1000* in each one.

4.2 Registration and call set-up

The WebSocket frames recorded in the browser show the sequence of Figure 4. Authentication is the Digest exchange of section 2.4 with `qop=auth`; the INVITE is challenged a second time because Asterisk authenticates the caller before entering the dialplan.

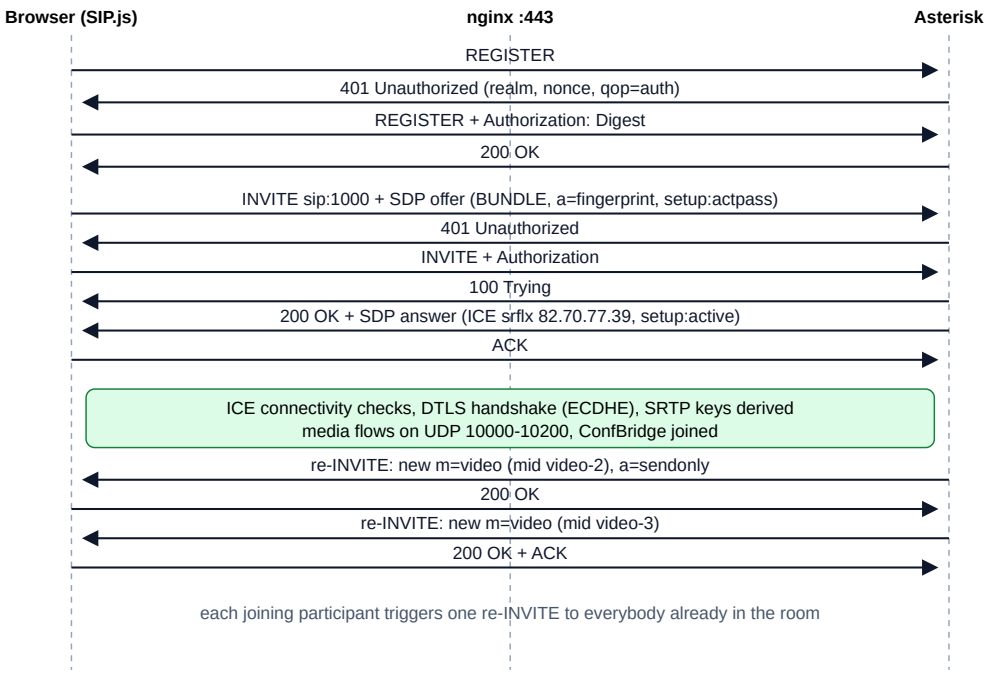


Figure 4. Signalling sequence for one participant joining room 1000.

Excerpt of the offer sent by a browser (the INVITE to 1000, abridged):

```
INVITE sip:1000@web-conference-lar0053.duckdns.org SIP/2.0
Via: SIP/2.0/WSS 0r60m72pnqc6.invalid;branch=z9hG4bK2437454
To: <sip:1000@web-conference-lar0053.duckdns.org>
From: "3042" <sip:3042@web-conference-lar0053.duckdns.org>;tag=fqlqvoq1s9
CSeq: 2 INVITE
Authorization: Digest algorithm=MD5, username="3042", realm="asterisk", nonce="1794073406/c89343404
7cdba5c629e14c170754ea6",    uri="sip:1000@web-conference-lar0053.duckdns.org",    response="fbafb75
a...", opaque="69dff7062811317a", qop=auth, cnonce="41u4crnv9nnp", nc=00000001
a=group:BUNDLE 0 1
m=audio 58050 UDP/TLS/RTP/SAVPF 111 63 9 0 8 13 110 126
a=candidate:3018048475 1 udp 2122194687 100.71.190.53 58050 typ host
a=candidate:3350141626 1 udp 2122063615 158.196.230.193 43929 typ host
a=ice-ufrag:KPIj
a=fingerprint:sha-256 85:A1:38:D6:A5:63:62:3E:...
a=setup:actpass
a=mid:0
a=sendrecv
a=rtcp-mux
a=rtpmap:111 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 9 UDP/TLS/RTP/SAVPF 96 97 103 ...
a=ice-ufrag:KPIj
a=fingerprint:sha-256 85:A1:38:D6:A5:63:62:3E:...
a=setup:actpass
a=mid:1
a=sendrecv
a=rtcp-mux
a=rtpmap:96 VP8/90000
a=rtpmap:97 rtx/90000
a=rtpmap:103 H264/90000
```

Asterisk's answer picks Opus, PCMU and PCMA for audio and VP8 and H.264 for video, becomes the DTLS client (`setup:active`) and publishes both its private and its public address as candidates:

```
SIP/2.0 200 OK
CSeq: 2 INVITE
a=group:BUNDLE 0 1
m=audio 10176 UDP/TLS/RTP/SAVPF 111 0 8 126
a=setup:active
a=fingerprint:SHA-256 40:0B:A3:B7:21:26:4D:30:...
a=ice-ufrag:6530d8a1...
a=candidate:Ha000058 1 UDP 2130706431 10.0.0.88 10176 typ host
a=candidate:H8aad688 1 UDP 2130706431 fe80::17ff:fe00:4a0c 10176 typ host
a=candidate:S52464d27 1 UDP 1694498815 82.70.77.39 10176 typ srflx raddr 10.0.0.88 rport 10176
a=rtpmap:111 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:126 telephone-event/8000
a=sendrecv
a=mid:0
m=video 10176 UDP/TLS/RTP/SAVPF 96 103
a=rtpmap:96 VP8/90000
a=rtpmap:103 H264/90000
a=sendrecv
a=mid:1
```

When the third, fourth and fifth participants join, Asterisk sends a re-INVITE to the people already connected. Each one adds a `sendonly` `m=video` section, with its own `mid` and `msid`, for the new participant. In the capture, the number of video sections in successive re-INVITEs received by 3042 was 2, 3, 4 and 5. This is the Unified Plan behaviour the SFU mode relies on, and SIP.js handled it without changes.

```
INVITE sip:gh1ai9d9@127.0.0.1:40206;transport=WS;ob SIP/2.0
CSeq: 21202 INVITE
m=audio 10176 UDP/TLS/RTP/SAVPF 111 0 8 126
a=setup:actpass
a=sendrecv
a=msid:493aaad0... ...
a=mid:0
m=video 10176 UDP/TLS/RTP/SAVPF 96 103
a=setup:actpass
a=sendrecv
a=msid:493aaad0... ...
a=mid:1
m=video 0 UDP/TLS/RTP/SAVPF 0
a=mid:video-2
m=video 0 UDP/TLS/RTP/SAVPF 96 103
a=mid:video-3
m=video 0 UDP/TLS/RTP/SAVPF 96 103
a=mid:video-4
m=video 0 UDP/TLS/RTP/SAVPF 96 103
a=mid:video-5
```

4.3 Server state during the conference

The following output was taken from the live server with five participants in the room.

```
$ asterisk -rx "core show version"
Asterisk 20.6.0~dfsg+~cs6.13.40431414-2build5 built by nobody @ buildd.debian.org on a unknown runn
ing Linux on 2024-04-15 00:13:12 UTC

$ asterisk -rx "http show status"
HTTP Server Status:
Prefix:
Server: Asterisk-VoIP1
Server Enabled and Bound to 0.0.0.0:8088

HTTPS Server Enabled and Bound to 0.0.0.0:8089

Enabled URI's:
/httpstatus => Asterisk HTTP General Status
/phoneprov/... => Asterisk HTTP Phone Provisioning Tool
/metrics/... => Prometheus Metrics URI
/ari/... => Asterisk RESTful API
/ws => Asterisk HTTP WebSocket

Enabled Redirects:
  None.

$ asterisk -rx "confbridge show profile bridge sfu_video_bridge"
------------------------------------------
Name:                 sfu_video_bridge
Language:             en
Internal Sample Rate: auto
Maximum Sample Rate: none
Mixing Interval:      20
Record Conference:    no
Record File Append:    yes
Record File Timestamp: yes
Record File:          Auto Generated
Record Options:
Record Command:
Max Members:          10
Registration context:
Video Mode:           sfu
Video Update Discard: 2000
REMB Send Interval: 0
```

```
$ asterisk -rx "pjsip show endpoints"
 Endpoint:  <Endpoint/CID.....................................>  <State.....>  <Channels.>
    I/OAuth:  <AuthId/UserName.....................................................>
        Aor:  <Aor.........................................>  <MaxContact>
     Contact:  <Aor/ContactUri.........................> <Hash....> <Status> <RTT(ms)..>
  Transport:  <TransportId........>  <Type>  <cos>  <tos>  <BindAddress.................>
   Identify:  <Identify/Endpoint.....................................................>
        Match:  <criteria.......................>
    Channel:  <ChannelId.....................................>  <State.....>  <Time.....>
        Exten: <DialedExten...........>  CLCID: <ConnectedLineCID.......>
==========================================================================================




 Endpoint:  3041/3041                                            In use        1 of inf
    InAuth:  3041/3041
       Aor:  3041                                               5
     Contact:  3041/sip:cg792o7v@127.0.0.1:39858;transpor 54deeab110 NonQual         nan
     Contact:  3041/sip:g5mppoco@127.0.0.1:35458;transpor d89ddb0e59 NonQual         nan
   Channel: PJSIP/3041-0000000d/ConfBridge                       Up            00:00:56
       Exten: 1000                     CLCID: "" <>

 Endpoint:  3042/3042                                            In use        1 of inf
    InAuth:  3042/3042
       Aor:  3042                                               5
     Contact:  3042/sip:1qs2jm5o@127.0.0.1:60644;transpor 331773ffca NonQual         nan
     Contact:  3042/sip:gh1ai9d9@127.0.0.1:40206;transpor 64a2f6bc3c NonQual         nan
   Channel: PJSIP/3042-0000000e/ConfBridge                       Up            00:00:51
       Exten: 1000                     CLCID: "" <>



 Endpoint:  Bolek/Bolek                                          In use        1 of inf
    InAuth:  Bolek/Bolek
       Aor:  Bolek                                              5
     Contact:  Bolek/sip:16k1dj5i@127.0.0.1:52048;transpo 6aba17a553 NonQual         nan
     Contact:  Bolek/sip:p8erpp5r@127.0.0.1:33828;transpo bc0efe7c1d NonQual         nan
   Channel: PJSIP/Bolek-00000010/ConfBridge                      Up            00:00:43
       Exten: 1000                     CLCID: "" <>




...

$ asterisk -rx "confbridge list 1000"
Channel                         Flags  User Profile     Bridge Profile   Menu             CallerID
=============================== ====== ================ ================ ================ ==========
======
PJSIP/3041-0000000d                    default_user     sfu_video_bridge default_menu     3041
PJSIP/3042-0000000e                    default_user     sfu_video_bridge default_menu     3042
PJSIP/3043-0000000f                    default_user     sfu_video_bridge default_menu     3043
PJSIP/Bolek-00000010                   default_user     sfu_video_bridge default_menu     Bolek
PJSIP/Lolek-00000011                   default_user     sfu_video_bridge default_menu     Lolek

$ asterisk -rx "pjsip show channelstats"
                                                        ...........Receive......... .........Transmi
```

```
t.........
 BridgeId ChannelId ........ UpTime.. Codec.   Count    Lost Pct  Jitter   Count    Lost Pct  Jitte
r RTT....
  ================================================================================================
=========

 1e88c100 3041-0000000d      00:00:56 opus     2748       0    0   0.001   2746       0    0   0.00
3   0.059
 1e88c100 3042-0000000e      00:00:51 opus     2523       0    0   0.004   2522       0    0   0.00
3   0.068
 1e88c100 3043-0000000f      00:00:47 opus     2304       0    0   0.001   2303       0    0   0.00
3   0.068
 1e88c100 Bolek-00000010     00:00:43 opus     2114       0    0   0.001   2113       0    0   0.00
2   0.063
 1e88c100 Lolek-00000011     00:00:38 opus     1889       0    0   0.001   1888       0    0   0.00
3   0.061

Objects found: 5
```

```
$ asterisk -rx "core show channels"
Channel                                              Location                       S
tate   Application(Data)
PJSIP/3042-0000000e                                  1000@default:2                 U
p      ConfBridge(1000,sfu_video_brid
PJSIP/3041-0000000d                                  1000@default:2                 U
p      ConfBridge(1000,sfu_video_brid
PJSIP/Bolek-00000010                                 1000@default:2                 U
p      ConfBridge(1000,sfu_video_brid
PJSIP/Lolek-00000011                                 1000@default:2                 U
p      ConfBridge(1000,sfu_video_brid
CBAnn/1000-00000002;2                                s@default:1                    U
p      (None)
CBAnn/1000-00000002;1                                s@default:1                    U
p      (None)
PJSIP/3043-0000000f                                  1000@default:2                 U
p      ConfBridge(1000,sfu_video_brid
7 active channels
5 active calls
18 calls processed

$ asterisk -rx "pjsip show aors"
      Aor:  <Aor.........................................>  <MaxContact>
    Contact:  <Aor/ContactUri.........................> <Hash....> <Status> <RTT(ms)..>
===============================================================================================

      Aor:  3001                                     3

      Aor:  3002                                     3

      Aor:  3003                                     3

      Aor:  3041                                     5
    Contact:  3041/sip:cg792o7v@127.0.0.1:39858;transport= 54deeab110 NonQual         nan
    Contact:  3041/sip:g5mppoco@127.0.0.1:35458;transport= d89ddb0e59 NonQual         nan
```

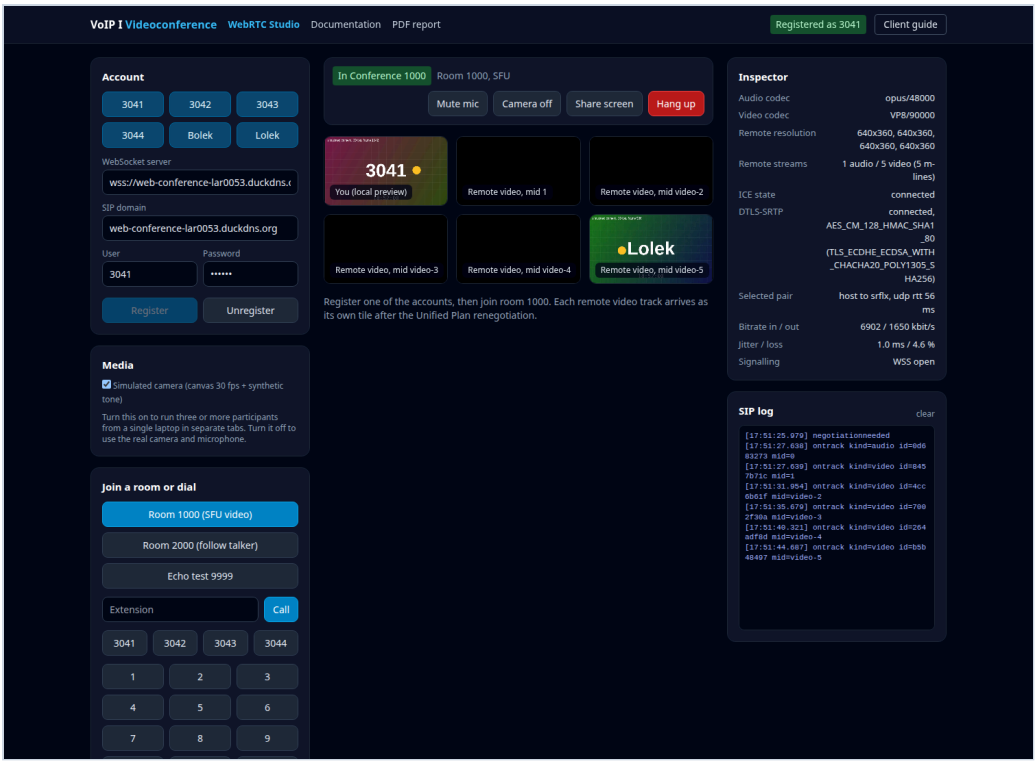


Figure 5. The studio as seen by participant 3041 in the five-party test.

4.4 Bandwidth analysis

The bitrate of one media stream is its payload rate plus the headers that every packet carries. For SRTP over IPv4/UDP the overhead per packet is 20 bytes (IP), 8 (UDP), 12 (RTP) and 10 (HMAC-SHA1-80 tag), 50 bytes in total. With 20 ms packetisation that is 50 packets per second, so the header cost is $50 \times 50 \times 8 = 20$ kbit/s per audio stream.

Stream	Payload	Packets/s	Overhead	On the wire
Opus, 24 kbit/s speech	24 kbit/s	50	20 kbit/s	44 kbit/s
G.711 (PCMU/PCMA), 160 B per 20 ms	64 kbit/s	50	20 kbit/s	84 kbit/s
VP8 360p, typical	700 kbit/s	about 75	about 30 kbit/s	about 730 kbit/s
VP8 720p, typical	1500 kbit/s	about 160	about 64 kbit/s	about 1560 kbit/s

One participant therefore produces about 0.77 Mbit/s at 360p (video plus Opus) and about 1.6 Mbit/s at 720p. Applying the SFU formulas of section 2.1 with these values:

N	SFU, 360p (Mbit/s)			SFU, 720p (Mbit/s)			Mesh 360p up / down
	client up	client down	server out	client up	client down	server out	
3	0.77	1.54	4.6	1.60	3.20	9.6	1.54 / 1.54
4	0.77	2.31	9.2	1.60	4.80	19.2	2.31 / 2.31
5	0.77	3.08	15.4	1.60	6.40	32.0	3.08 / 3.08

Table: expected traffic per client and at the server, with b = 0.77 Mbit/s (360p plus Opus) and 1.60 Mbit/s (720p plus Opus).

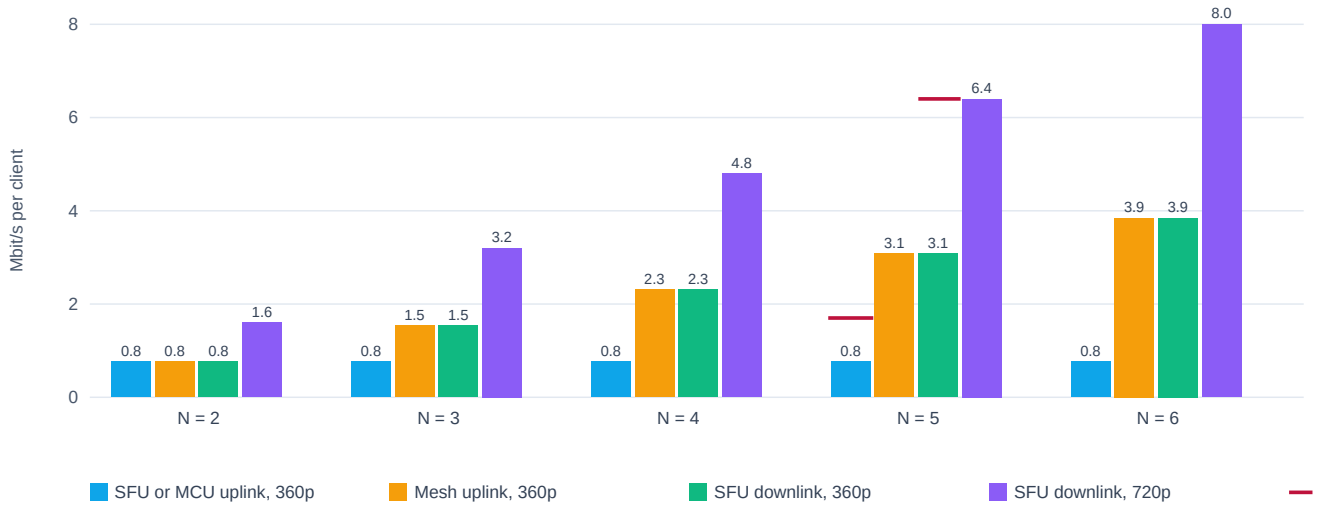


Figure 6. Expected bandwidth per client by topology. The red marks are the values measured with five participants (about 1.7 Mbit/s up, 6.4 Mbit/s down, synthetic canvas video).

The measured values agree with the model. During the five-party run each client uploaded between 1.4 and 1.8 Mbit/s and received between 6.2 and 6.6 Mbit/s, a downlink-to-uplink ratio of 3.6 to 4.4, close to the expected N−1 = 4. The absolute rate is about twice the typical 360p figure because the synthetic canvas is a worst case for VP8: the moving gradient and grid change every pixel on every frame, and Chrome's default encoder target for 640×360 is above the "typical" 700 kbit/s. Real camera video of a talking head would use less.

Quality measurements from the browsers' `getStats()` and from `pjsip show channelstats` on the server (five participants, about one minute):

Metric	Value	Source
Round-trip time to server	56 to 79 ms	ICE candidate pair, browser
Audio jitter	0 to 5 ms	browser inbound-rtp
Audio jitter (server view)	1 to 4 ms	channelstats
Audio packet loss on the server	0 of about 2000 packets	channelstats
Packet loss seen by the browsers	1.3 to 3.5 %	browser inbound-rtp (audio and video)
Selected ICE pair	host to srflx, UDP	browser
Negotiated media	Opus 48 kHz, VP8, DTLS-SRTP (AES_CM_128_HMAC_SHA1_80)	browser

Loss of one to three percent is visible only in the browser measurements; the server counted none on audio. All five browser sessions ran on one laptop and decoded up to four 360p streams each, so the loss includes receive-buffer drops caused by local CPU load and should not be read as network quality. A real deployment with one participant per machine would need its own measurement.

4.5 Other rooms and clients

Room 2000 uses `follow_talker`. Three participants (3041, 3042, 3043) joined it and each browser received one audio and one video stream (640×360 VP8, Opus), where the same three clients in room 1000 receive one video section per other participant. That is the expected difference: the bridge switches a single video stream to whoever speaks. Extension 9999 was tested with Bolek: audio and video came back from the server over DTLS-SRTP, ICE connected. Direct calls between WebRTC accounts use `Dial(PJSIP/${EXTEN})` and need no conference; I did not run a separate test for them. Classic

accounts 3001 to 3003 are configured for UDP SIP without media encryption; they were not tested with a hardware or desktop softphone during this project, and neither was sipML5.

5. Own opinion and contribution

5.1 ConfBridge SFU compared with dedicated SFUs

	Asterisk ConfBridge SFU	**Mediasoup**	**Janus VideoRoom**	**Jitsi Videobridge**	**LiveKit**
Language	C	C++ worker, Node API	C	Java	Go
Simulcast / SVC	no	yes	yes	yes	yes
SIP and PSTN integration	native	none	plugin	via Jigasi	SIP service
Audio mixing for phones	yes (mixed audio)	no	optional	no	no
Browser signalling	SIP over WS, re-INVITE	custom	JSON plugin API	XMPP / Colibri	own protocol
Scaling	single node	multi-worker	single node, clusterable	horizontal	horizontal

My view after building it: ConfBridge is the right tool when the video conference has to be an extension of a PBX. The same room answers a browser, a SIP desk phone and a PSTN call, the audio is mixed on the server so a phone that cannot handle several streams still hears everybody, and the configuration is a few dozen lines. It is the wrong tool when the product is video: it has no simulcast, so every receiver gets every sender at full quality and a participant with a weak connection degrades the experience of that connection only by losing packets, not by getting a smaller picture. Mediasoup, Janus and LiveKit solve that and also give the application full control of which streams each client receives. For a university exercise, and for a small team that already runs Asterisk, ConfBridge is enough.

5.2 Problems found and how they were solved

- **chan_sip took the WebSocket.** The Debian package loads both SIP drivers. chan_sip registered the `sip` sub-protocol first, so the WSS handshake succeeded but every REGISTER was handled by the old driver and failed with 403 InvalidPassword in `security_log`, even though the credentials were right. A hand-written Digest client confirmed that the server side was at fault. Adding `noload => chan_sip.so` under `[modules]` (not under `[global]`, where the first attempt put it) and restarting fixed it; `res_pjsip_transport_websocket` only started running after that restart.
- **NAT in the cloud.** The VM sees only 10.0.0.88. With `external_media_address`, `external_signaling_address` and `local_net` in every transport, plus ICE and a STUN server in `rtp.conf`, the answer carries the public address as a srflx candidate and the clients connected on the first attempt over UDP. STUN to the IPv6 address of Google's server logs "Network is unreachable" because the VM has no IPv6; it is harmless.
- **WebSocket on 443 instead of 8089.** Port 8089 is not open in the Oracle security list, and many corporate networks only allow 443. Proxying `/ws` through nginx removes the need to open anything for signalling and reuses the existing certificate. The 8089 listener still works on the host (checked with a handshake against 127.0.0.1) but is not reachable from outside until the cloud rule exists.
- **Renegotiation as people join.** Each new participant makes Asterisk re-INVITE everybody with one more `sendonly` video section. SIP.js accepted every one automatically, so the work was in the page: creating a tile per track and removing it on `ended`. One side effect: some video sections carry no frames (the original sendrecv section in the SFU, and sections of participants who have left), so in the five-party run each browser showed four decoded streams

out of five video sections. The page creates a tile for every track and hides it while the track is muted, but the inspector counts m-lines and decoded streams separately, and a tile can still stay empty for a moment after a renegotiation.

- **`rtp_timeout` breaks the SFU.** I first set `rtp_timeout=30` to clean up channels left behind when a browser closes without a BYE. Asterisk then hung up all five participants after 30 seconds with "lack of video RTP activity", because some of the extra video sections legitimately carry no packets. The option was removed. Stale channels are removed with `channel request hangup` or when the registration expires; a better fix would be an AMI watcher.
- **One camera, many participants.** A physical camera can be opened by one process, which makes a three-tab demonstration impossible. The canvas stream described in section 3.4 removes that limit and makes the test reproducible, and it also made automated testing with headless Chrome possible.
- **Opus in a mixing bridge.** The open-source Opus module in the Debian package is a pass-through codec with limited transcoding. It worked here because the browsers negotiated Opus, ConfBridge handled it, and the server-side channelstats show Opus on all five channels. I did not test mixed Opus and G.711 participants in one room, and a deployment relying on that should be tested.

5.3 Contribution

The parts that are my own work, beyond configuring the packages: a single-file browser studio that exposes the negotiation (SIP log, codecs, ICE, DTLS, bitrate) and works with one laptop; the headless test harness that registers five accounts, joins the room and records the WebSocket frames and statistics, which produced all the traces in this report; the bandwidth model compared with measurements; the automated certificate hook that copies renewals into Asterisk; and a fail2ban jail on the Asterisk security log. This report and the portal are generated from the same source so they cannot drift apart.

5.4 Security notes

The accounts use short, guessable passwords because they are demo credentials, and the server is on the public Internet. Three measures reduce the risk: fail2ban watches `/var/log/asterisk/security_log` and bans an address after ten failed attempts in ten minutes for fifteen minutes; signalling travels only over WSS; and media is encrypted with DTLS-SRTP. This is not enough for production. The passwords should be long random strings, the SSH jail was deliberately left disabled so the existing access to the machine did not change, and the classic SIP accounts on UDP 5060 are exposed to scanning and should be restricted by ACL or removed after the demonstration. No TURN server is deployed.

6. Conclusion

A conference with five simultaneous participants, each seeing four other video streams and hearing a mixed audio stream, runs on an open-source stack on a four-core ARM server, with signalling over WSS on port 443 and encrypted media over a range of 201 UDP ports. The measured traffic matches the SFU model: one upload and N−1 downloads per client, about 1.5 Mbit/s up and 6.4 Mbit/s down in the worst-case synthetic video. Round-trip time to the server was 56 to 79 ms and audio jitter a few milliseconds.

The weak points are known and documented: no simulcast in ConfBridge, no TURN fallback, demo-grade passwords, a cloud firewall that must be opened by hand for the optional ports, and softphone clients (Linphone, MicroSIP, sipML5) that are configured but were not verified. The next steps would be to add a TURN server, to test a real mix of browsers and softphones, and to compare ConfBridge against Janus or LiveKit under the same load.

7. References

1. J. Rosenberg et al., "SIP: Session Initiation Protocol," RFC 3261, IETF, June 2002.
2. M. Baugher et al., "The Secure Real-time Transport Protocol (SRTP)," RFC 3711, IETF, March 2004.
3. F. Andreasen, M. Baugher and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams," RFC 4568, IETF, July 2006.
4. D. McGrew and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)," RFC 5764, IETF, May 2010.

5. P. Zimmermann, A. Johnston and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP," RFC 6189, IETF, April 2011.
6. I. Baz Castillo, J. Millan Villegas and V. Pascual, "The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)," RFC 7118, IETF, January 2014.
7. A. Keranen, C. Holmberg and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," RFC 8445, IETF, July 2018.
8. A. Begen, P. Kyzivat, C. Perkins and M. Handley, "SDP: Session Description Protocol," RFC 8866, IETF, January 2021.
9. J. Uberti, C. Jennings and E. Rescorla, "Javascript Session Establishment Protocol (JSEP)," RFC 8829, IETF, January 2021.
10. Asterisk Project, "Asterisk 20 LTS documentation: PJSIP configuration, ConfBridge and WebRTC," Sangoma Technologies, https://docs.asterisk.org, accessed November 2026.
11. J. Garcia et al., SIP.js 0.21 documentation, https://sipjs.com, accessed November 2026.
12. F. Řezáč, lecture notes "Voice over IP I", lectures 7, 8, 9 and 13, VŠB-TU Ostrava, 2026.
13. M. Vozňák, *Voice over IP* course material, VŠB-TU Ostrava.