The Galène videoconferencing server

Juliusz Chroboczek

22 February 2021
Galène is a videoconferencing server.

https://galene.org

You’re using it now.
Galène UI

**UI designed for lectures:**
- you start with the camera/microphone off, switch on with *Ready*;
- you can share multiple windows;
- breakout groups created automatically;
- video **mirroring** (mirror in front of webcam).

**Compromises were made:**
- you can **play videos** (watching videos during lockdown) (currently broken in Chrome/Chromium);
- can **switch camera on at login** (apparently essential for meetings).
Galène UI (2)

UI still incomplete:
- some functions are only available as commands
  - type /help in the chat
  - students love /msg;
  - administrator has extra commands (/kick, /mute);
- only one layout for now
  - use full screen;
  - use picture-in-picture.

Accepted by students, but not by non-CS lecturers.
(Some administrators hate it!)
Galène is a videoconferencing server:
- designed for teaching, but useful for meetings;
- easy to compile and deploy
  (15 minutes according to Dave);
- small: 7000 lines of Go and 4000 lines of JS;
- minimal server resources:
  - 5€/month VPS for 100-person lectures;
  - runs fine on a 50€ ARM board;
  - multicore scaling
    (teaching is eternally underfunded);
- can run off a read-only filesystem.
The first French lockdown

First French lockdown:
- 17 March through 11 May 2020;
- stuck in 40 $m^2$
  (Mayor of Paris forbids public parks!);
- working on mobility in IP networks
  (no motivation, cannot test);
- need to lecture from home.

Contradictory orders:
- University buys a Zoom licence
  (for an undisclosed sum);
- national CNRS forbids Zoom;
- local CNRS uses Zoom.

But some of us prefer self-hosted software!
The first French lockdown (2)

Need to lecture from home.
- University buys a Zoom licence;
- some of us prefer self-hosted software.

Finally, a self-hosted instance of BigBlueButton:
- often overloaded;
- only allowed for work-related purposes.

BigBlueButton:
- full of features;
- requires serious server-side resources.
The first French lockdown (3)

BigBlueButton:
   – full of features (great for teaching!);
   – requires serious server-side resources.

At that point, I should have:
   – installed an instance of Jitsi; or
   – written a frontent for Janus; or
   – looked in more detail at Ion.

But I didn’t. I wrote my own.

Never do that!

(I mean, seriously, don’t.)
Videoconferencing is difficult

At first sight, videoconferencing is difficult:

- **signalling:**
  - codec negotiation;
  - NAT traversal;

- **media flow distribution:**
  - loss recovery;
  - congestion control;

- **video quality:**
  - jitter compensation;
  - lipsynch;

- **video and audio codecs**
  - oh my!
Videoconferencing made tolerable: WebRTC

At first sight, videoconferencing is difficult.

It has recently become tolerable: WebRTC:
  – a complete videoconferencing stack;
  – implemented in major browsers;
  – finally agreed on common codecs:
    – everyone implements Opus and VP8 (even Apple!).
WebRTC: a peer-to-peer protocol

WebRTC implemented in the browser:

- client-server signalling (WebSocket, REST);
- media is peer-to-peer (RTP+RTCP+SRTP);
- optional peer-to-peer data (SCTP+DTLS).

The media traffic is encrypted end-to-end, keys negotiated over the signalling channel.

The JavaScript API is simple but inflexible (leading to “SDP munging”) (where art thou, ORTC?).
Peer-to-peer WebRTC doesn’t scale

What happens if you try to broadcast over WebRTC?

Every p2p flow is encoded, encrypted and sent separately.

 Doesn’t scale beyond 4 or 5 peers.
Client-server WebRTC

The solution is **client-server**:  
– client-server signalling;  
– **client-server media**.

No need to reencode the media.

The server **decrypts** and **reencrypts** the video: there is **no end-to-end confidentiality**.

(Yes, I know about insertable streams.)
In the peer-to-peer case, WebRTC is in the browser.

Client-server, you need server-side WebRTC:
- RTP and RTCP;
- SRTP;
- SCTP;
- STUN, TURN, ICE...

**Pion** is a Go implementation of WebRTC:
- pure Go (easy to cross-compile);
- lower layers fairly complete, upper layers in progress;
- reactive and friendly maintainer (Sean DuBois).

Galène uses **Pion**. Excellent experience.
Loss handling

Once you do client-server, where do you handle packet loss?

In Galène, we handle packet loss locally:

- reduces latency;
- requires buffering at the server.

This buffering does not cause bufferbloat: packets are forwarded or dropped, never queued; the buffer is only used to serve NACKs from the client.
Buffer management

Buffers in Galène: what size?

Packets are not queued, the size doesn’t matter much:

– if too small, we won’t be able to serve NACKs locally (we forward to the sender, increasing latency);
– if too large, we’ll waste memory.

Currently sized proportionally to

\[ \text{rate} \cdot (\text{maxRTT} + 4 \cdot \text{jitter}) \]

More experimentation is needed.
Congestion control

How fast can we send data over a given link? That’s the problem of congestion control.

WebRTC doesn’t define congestion control. Browsers implement Google Congestion Control (GCC), which combines two congestion controllers:

– a traditional loss-based controller (useless in the presence of bufferbloat);
– a novel delay-based controller.

In Galène, we terminate congestion control at the SFU. Galène acts as an application-layer proxy.
Congestion control (2)

We terminate congestion control at the SFU.

The resulting data rate is the minimum of the data rates acceptable for all clients:

– for small meetings, high rate;
– during large lectures, the rate falls down to the minimum.

Potential solutions: simulcast or SVC.

Right now, congestion control in Galène is incomplete:

– complete in the server → client direction;
– loss-based in the client → server direction.

Due to the prevalence of bufferbloated routers, this needs fixing.
Current status

Galène is good enough for lectures with 100 students:
- robust server (doesn’t crash or deadlock);
- robust NAT traversal (many students are on 4G) (thanks to Pion and ICE);
- robust loss recovery.

Congestion control:
- state of the art in the server→client direction (loss- and delay-based);
- loss-based in the client→server direction (requires manual tweaking on bufferbloated networks).
Current status(2)

With a fascist firewall, Galène keeps trying:
- difficult to determine when to give up;
- UI issue: how to indicate that there is a problem?

Good video quality:
- NACKs served locally in a timely manner;
- PLIs aggregated and forwarded to the sender.

Audio quality issues:
- browsers don’t implement (enough) audio FEC
  please implement flexfec in the browsers!
Future plans

Improve the UI:
- Ready/Panic is not obvious;
- multiple layouts;
- contextual menus and mouse-over text;
- alternate frontends?

Vary quality per client:
- simulcast;
- scalable video coding (SVC).

Improve congestion control:
- many networks are bufferbloated!
Conclusion

Galène is a videoconferencing server:
- easy to deploy;
- easy to understand and improve;
- requires minimal server resources.

https://galene.org

Please install your own instance!