GStreamer in VR devices manufacturing

Ivan Loskutov
Software Engineer

Meta

About me

- Worked on different low-level software: from embedded to OS development
- Since 2021 work for Meta on camera and sensor recording software

What we are doing

- Camera and sensor recording software for factory calibration and algorithm validation
  - Modern VR devices have many cameras and sensors
  - On factories and labs we record all sensors to file on device for offline processing
  - Recording to VRS file format
  - All internal software use this format for data processing
- Lab data recordings to train and evaluate CV algorithms

Privacy note: Recording software is internal tool and is not included in devices firmware

VRS file format

- optimized to record and playback streams of sensor data
- multiple streams of time-sorted records
- streams may contain Configuration, State and Data records
- supports huge file sizes

Open-source: https://github.com/facebookresearch/vrs

Why GStreamer

- Need for an extensible media framework for recording and streaming
- Flexibility to support different requests from the internal partners
- Existing solution
  - Not flexible enough, e.g. adding streaming requires to change architecture
  - To simplify extension we have implemented plugin system, which was not so mature as GStreamer plugins
- GStreamer
  - We can reuse common building block for creating different pipelines (e.g. streaming)
  - Mature plugin system simplifies new source plugin development
  - Internal partners can implement plugins

Challenges

- Streaming and recording of uncompressed camera data at nominal frame rate
  - Required bandwidth > 1GB/s for all cameras
  - Limited by internal storage and network bandwidth
  - Workarounds - different configurations:
    - record only one set cameras for one case and another set for another case
    - reducing FPS
    - image binning
    - encoding
- Compatibility with existing tools and frameworks (data types, VRS)
  - Vrs-sink element
  - Streaming using internal data formats instead of raw/video-x
- Camera stack is different from other OS
  - All source plugins are developed in-house
- Support not only video and audio
- CPU load and memory usage limitation

How we use GStreamer

- RTSP solution for streaming
- On-device recorder
- Element library to build custom pipelines using gst-launch or programmatically

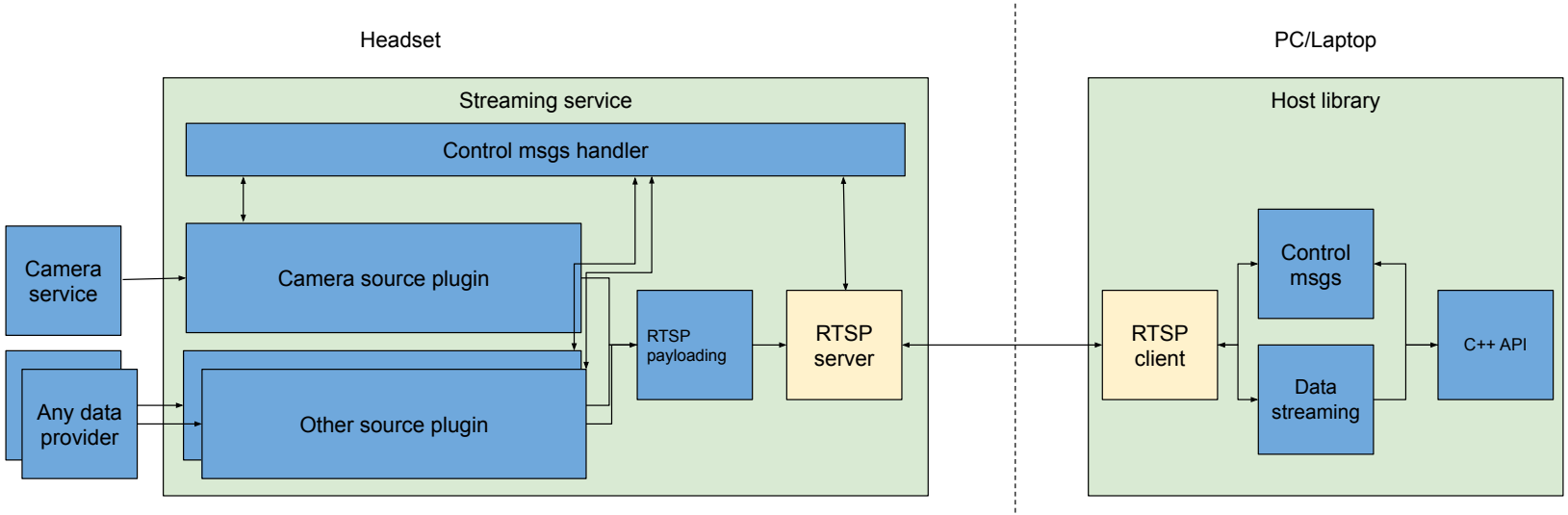
Streaming solution

- Server
  - Based on gstreamer-rtsp-server
  - Dynamic pipeline building and configuration
  - Custom control command for configuration and getting information using RTCP SetParameter and GetParameter.
- C++ client API library
  - simplifies integration to existing projects
    - use in-house data types
    - other teams don’t want interact with GStreamer directly

Why RTSP?

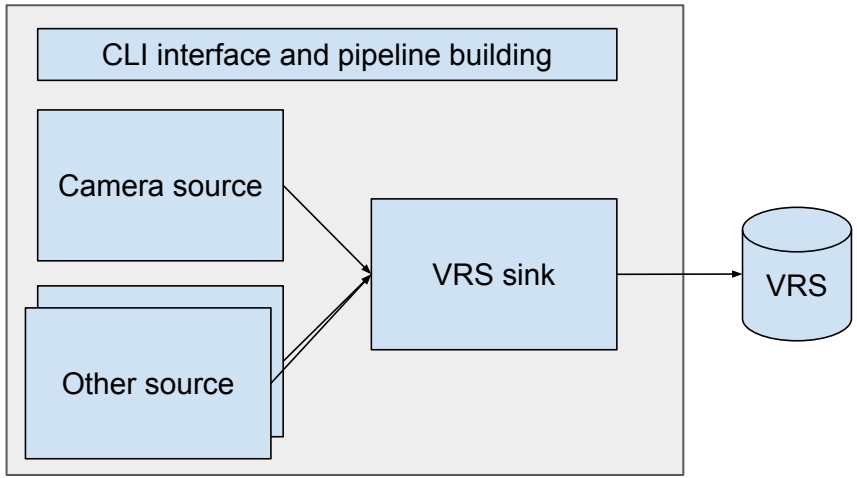
- Standardized protocol for streaming
- Supports play, stop, configuration commands out of the box
- Mature server and client implementations in GStreamer

Streaming solution



On device recorder

- Common code base with streaming solution
- CLI user interface for common recording cases
- Programmatically builds pipeline
- Produces VRS file



Custom pipelines

- For experimentation

```
gst-launch-1.0 \
  camerasrc <parameters> ! tee name=t \
  t. ! queue ! vrssink location=test.vrs \
  t. ! queue ! x264enc ! rtph264pay ! udpsink <params>
```

Challenges and learnings

- Ref-counting is difficult
  - GST Debug and Tracer
- Mixing with C++ code
  - Sanitizers and GST Leak tracer
  - Wrappers
- Steep learning curve
  - Many really good examples, but they cover mostly simple cases
  - Raising expertise in the team in the process of building products

Future plans

- Fully replace current solutions with GStreamer-based one
- Extend set of elements to cover all required recording and streaming cases
- Keep the new solution easily configurable and extendable to support more cases and simplify experimentation

Q&A


Meta