

Building a Ringbuffer

Going in circles with V4L2 and PipeWire

Simon Himmelbauer



Made In Austria 

Table of Contents

Introduction and Use Case

Implementation of Ringbuffer

Ringbuffer x PipeWire

About us

- Software Engineer @ A&R TECH

About us

- Software Engineer @ A&R TECH
- Mostly writing userspace stuff

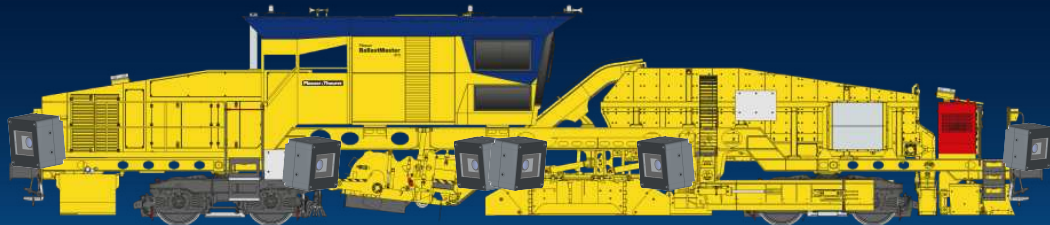
About us

- Software Engineer @ A&R TECH
- Mostly writing userspace stuff
- Developing video solutions for railway

About us

- Software Engineer @ A&R TECH
- Mostly writing userspace stuff
- Developing video solutions for railway
 - HMIs
 - SIL2 Cameras

Example Deployment



Hardware

- AMD Zynq UltraScale+
- Quad-core Arm Cortex-A53
- Dual-core Arm Cortex-R5

Hardware

- AMD Zynq UltraScale+
- Quad-core Arm Cortex-A53
- Dual-core Arm Cortex-R5

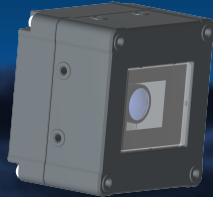


Table of Contents

Introduction and Use Case

Implementation of Ringbuffer

Ringbuffer x PipeWire

Building a Ringbuffer Element

- Derived from `GstBaseTransform`

Building a Ringbuffer Element

- Derived from `GstBaseTransform`
- Elements kept in a fixed size array

Building a Ringbuffer Element

- Derived from `GstBaseTransform`
- Elements kept in a fixed size array
- `*_propose_allocation` requests configured buffer pool size

Building a Ringbuffer Element

- Derived from `GstBaseTransform`
- Elements kept in a fixed size array
- `*_propose_allocation` requests configured buffer pool size
- `*_prepare_output_buffer` inserts buffer into array

Building a Ringbuffer Element

- Derived from `GstBaseTransform`
- Elements kept in a fixed size array
- `*_propose_allocation` requests configured buffer pool size
- `*_prepare_output_buffer` inserts buffer into array
 - If full, oldest buffer is evicted
 - Sends empty buffer with flag `GST_BUFFER_FLAG_GAP`

Building a Ringbuffer Element

- Derived from `GstBaseTransform`
- Elements kept in a fixed size array
- `*_propose_allocation` requests configured buffer pool size
- `*_prepare_output_buffer` inserts buffer into array
 - If full, oldest buffer is evicted
 - Sends empty buffer with flag `GST_BUFFER_FLAG_GAP`
- `emit-buffer` action signal allows emitting a buffer based on timestamp

Ringbuffer x Video4Linux2

- High CPU load observed for larger ringbuffer sizes

Ringbuffer x Video4Linux2

sys/v4l2/gstv4l2object.c

```
gboolean gst_v4l2_object_decide_allocation (GstV4l2Object *obj, GstQuery *query) {  
    // ...  
    gst_v4l2_get_driver_min_buffers (obj);  
    /* We can't share our own pool, if it exceed V4L2 capacity */  
    if (min + obj->min_buffers + 1 > VIDEO_MAX_FRAME)  
        can_share_own_pool = FALSE;  
    // ...  
}
```

Ringbuffer x Video4Linux2

sys/v4l2/ext/videodev2.h

```
/*  
 * Common stuff for both V4L1 and V4L2  
 * Moved from videodev.h  
 */  
#define VIDEO_MAX_FRAME      32  
#define VIDEO_MAX_PLANES    8
```

Ringbuffer x Video4Linux2

sys/v4l2/ext/videodev2.h

```
/*  
 * Common stuff for both V4L1 and V4L2  
 * Moved from videodev.h  
 */  
#define VIDEO_MAX_FRAME      32  
#define VIDEO_MAX_PLANES    8
```

- **SOLUTION:** Increase VIDEO_MAX_FRAME to a sane value

Ringbuffer x Video4Linux2

sys/v4l2/ext/videodev2.h

```
/*  
 * Common stuff for both V4L1 and V4L2  
 * Moved from videodev.h  
 */  
#define VIDEO_MAX_FRAME      32  
#define VIDEO_MAX_PLANES    8
```

- **SOLUTION:** Increase VIDEO_MAX_FRAME to a sane value
- 32 → 2048

Table of Contents

Introduction and Use Case

Implementation of Ringbuffer

Ringbuffer x PipeWire

PipeWire

- Known as PulseAudio and JACK successor

PipeWire

- Known as PulseAudio and JACK successor
- Can handle video (V4L2, libcamera, ...) too!

PipeWire

- Known as PulseAudio and JACK successor
- Can handle video (V4L2, libcamera, ...) too!
- Manages V4L2 devices

PipeWire

- Known as PulseAudio and JACK successor
- Can handle video (V4L2, libcamera, ...) too!
- Manages V4L2 devices
 - Enables multiprocess access
 - All zero-copy thanks to dmabuf

Challenges

- VIDEO_MAX_FRAME limited to 32

Challenges

- VIDEO_MAX_FRAME limited to 32
 - Increase to 2048

Challenges

- VIDEO_MAX_FRAME limited to 32
 - Increase to 2048
- Buffers queued back despite being in use?

```
0:00:04.839617192 3135 3144 ERROR pipewiresrc gstpipewiresrc.c:735:dequeue_buffer:<pipewiresrc0> buffer  
0x7f541c003fd0 was not recycled
```

PipeWire Architecture

Disclaimer

I am completely new to the architecture of PipeWire. Information might be inaccurate.

¹<https://indico.freedesktop.org/event/11/contributions/463/>

PipeWire Architecture

Disclaimer

I am completely new to the architecture of PipeWire. Information might be inaccurate.

- Graph-based processing
 - Great overview by George Kiagiadakis from the 2025 GStreamer Conference¹

¹<https://indico.freedesktop.org/event/11/contributions/463/>

PipeWire Architecture

Disclaimer

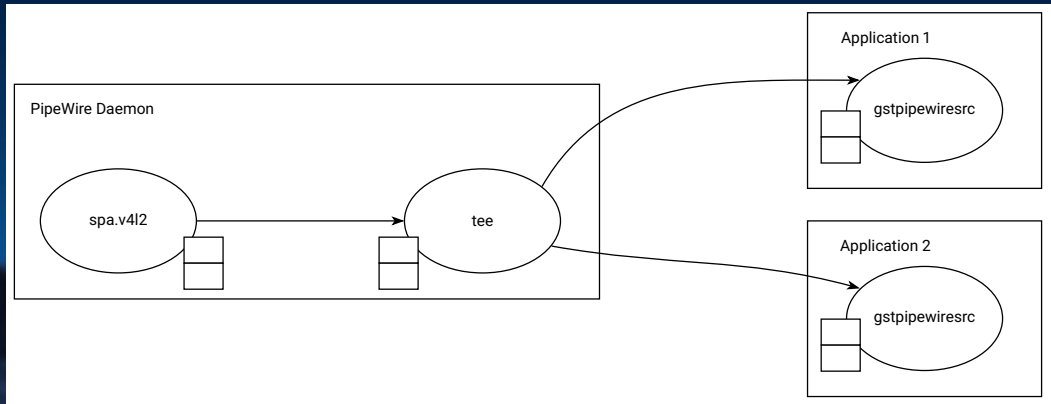
I am completely new to the architecture of PipeWire. Information might be inaccurate.

- Graph-based processing
 - Great overview by George Kiagiadakis from the 2025 GStreamer Conference¹
- Communication via `eventfd` and shared memory

¹<https://indico.freedesktop.org/event/11/contributions/463/>

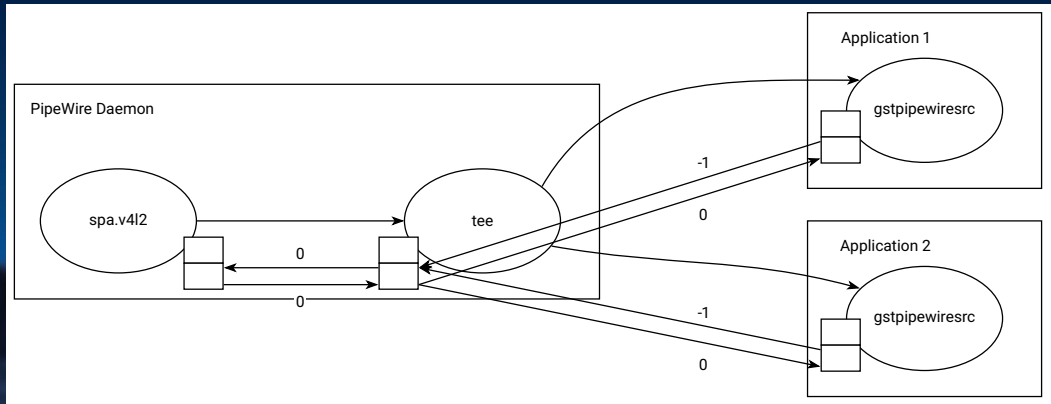
Async Graph Processing Example

Cycle 0



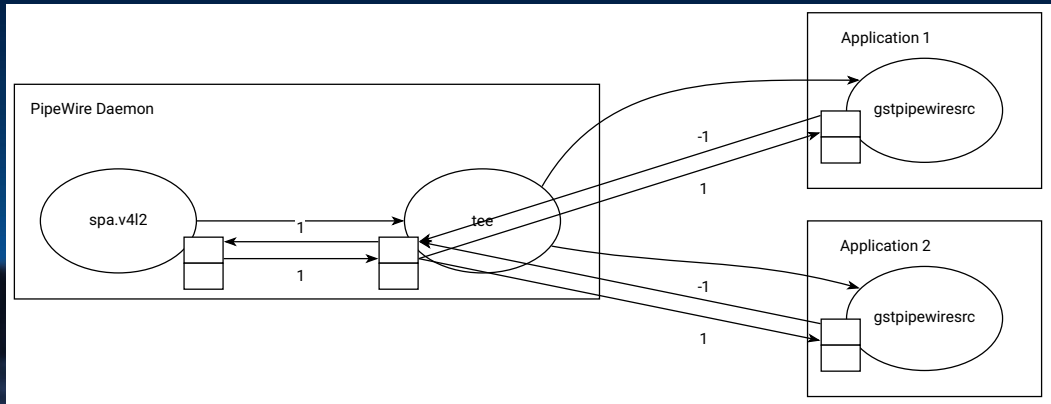
Async Graph Processing Example

Cycle 1



Async Graph Processing Example

Cycle 2



The Problem

- Buffer consumed by `tee` immediately returned

The Problem

- Buffer consumed by tee immediately returned
 - `spa.v4l2` will give buffer back to driver (`VIDIOC_QBUF`)
 - Assumes applications process buffer fast enough

The Problem

- Buffer consumed by tee immediately returned
 - `spa.v4l2` will give buffer back to driver (`VIDIOC_QBUF`)
 - Assumes applications process buffer fast enough
- No ownership tracking!

Attempted Approaches

- SPA_META_Busy adds atomic counter to buffers

Attempted Approaches

- SPA_META_Busy adds atomic counter to buffers
 - Used by `pw_stream`
 - Attempted to add support in `spa.v4l2`
 - Did not work due to race condition

Attempted Approaches

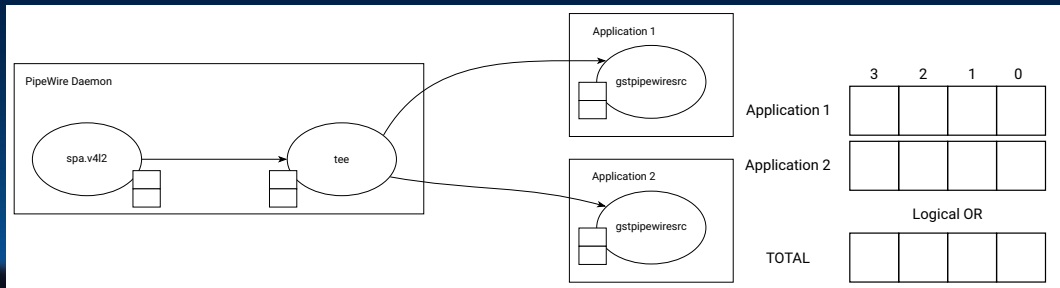
- SPA_META_Busy adds atomic counter to buffers
 - Used by `pw_stream`
 - Attempted to add support in `spa.v4l2`
 - Did not work due to race condition
- Handle buffer tracking in `tee`

Attempted Approaches

- SPA_META_Busy adds atomic counter to buffers
 - Used by `pw_stream`
 - Attempted to add support in `spa.v4l2`
 - Did not work due to race condition
- Handle buffer tracking in `tee`
 - Bitmask for each port to track usage
 - Only unused buffer returned to previous node
 - If impossible to reclaim buffer, drop current one

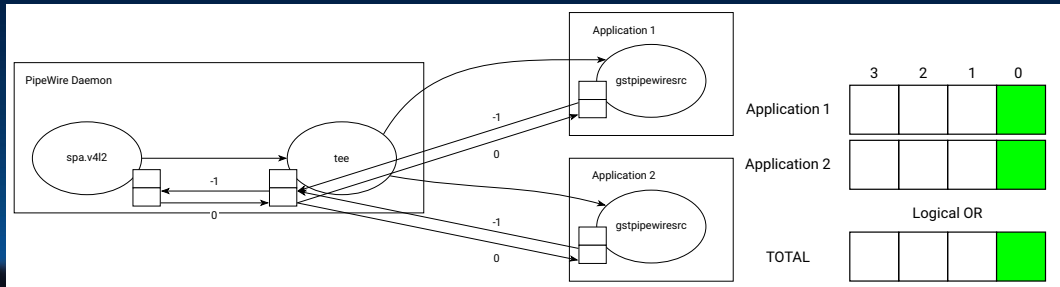
Async Graph Processing Example with Usage Tracking

Cycle 0



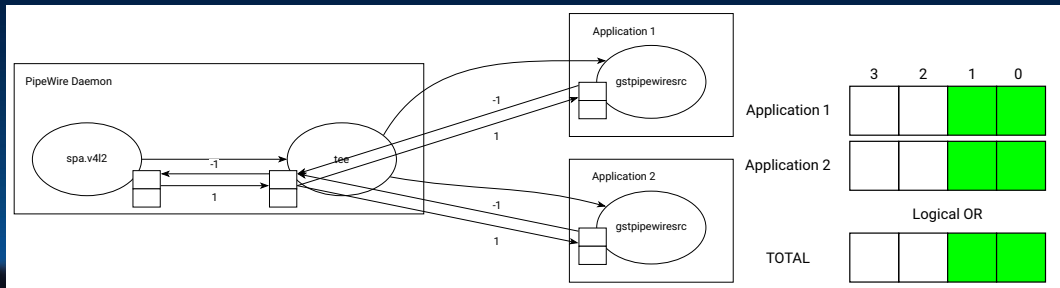
Async Graph Processing Example with Usage Tracking

Cycle 1



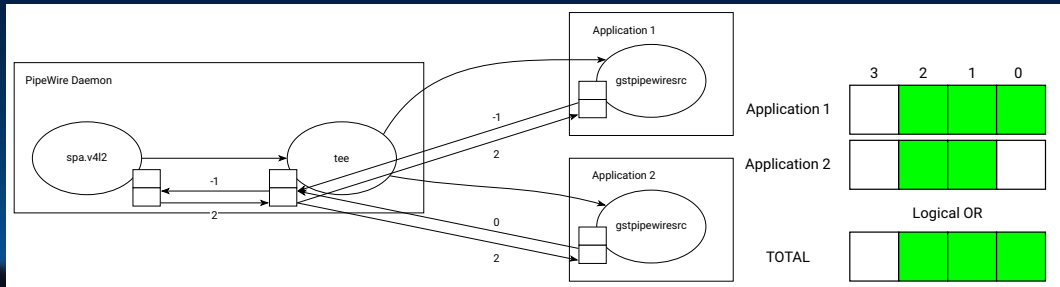
Async Graph Processing Example with Usage Tracking

Cycle 2



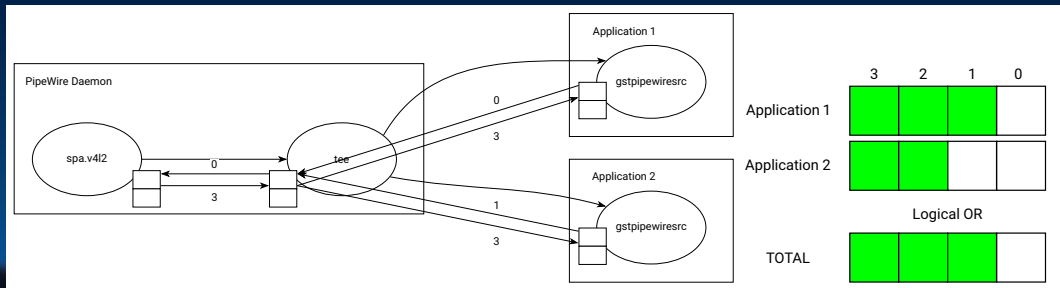
Async Graph Processing Example with Usage Tracking

Cycle 3



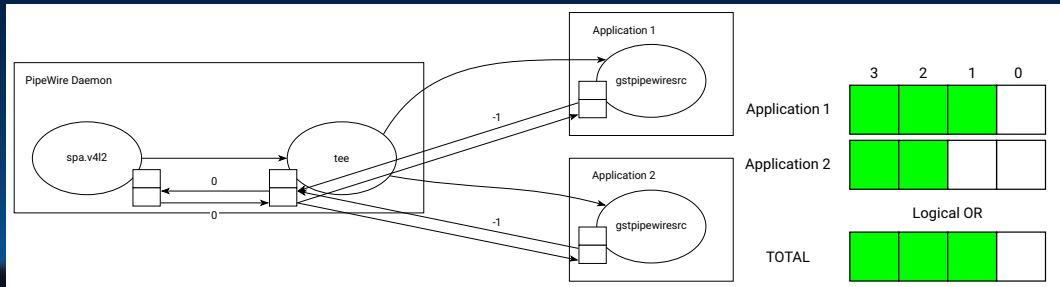
Async Graph Processing Example with Usage Tracking

Cycle 4



Async Graph Processing Example with Usage Tracking

Cycle 5



Conclusion

- It works (so far)!

Conclusion

- It works (so far)!
- Buffers only returned when returned by ALL consumers

Conclusion

- It works (so far)!
- Buffers only returned when returned by ALL consumers
- Source node NOT blocked when all buffers in use

Conclusion

- It works (so far)!
- Buffers only returned when returned by ALL consumers
- Source node NOT blocked when all buffers in use
- One consumer can prevent all other consumers from retrieving new buffers

Conclusion

- It works (so far)!
- Buffers only returned when returned by ALL consumers
- Source node NOT blocked when all buffers in use
- One consumer can prevent all other consumers from retrieving new buffers
- Shoutout to PipeWire people and their amazing help in Matrix chat

Questions?
