When adding more threads adds more problems

Thread-sharing between elements in GStreamer

GStreamer Conference 2018

25 October 2018, Edinburgh

Sebastian 'slomo' Dröge
< sebastian@centricular.com >

Who?

What?

Not about thread-safety!

but

Reducing the number of threads in a GStreamer pipeline

The Scenario

1. Receive an RTP audio stream
2. Transcode it
3. Send it out again
4. Do RTCP and jitterbuffer

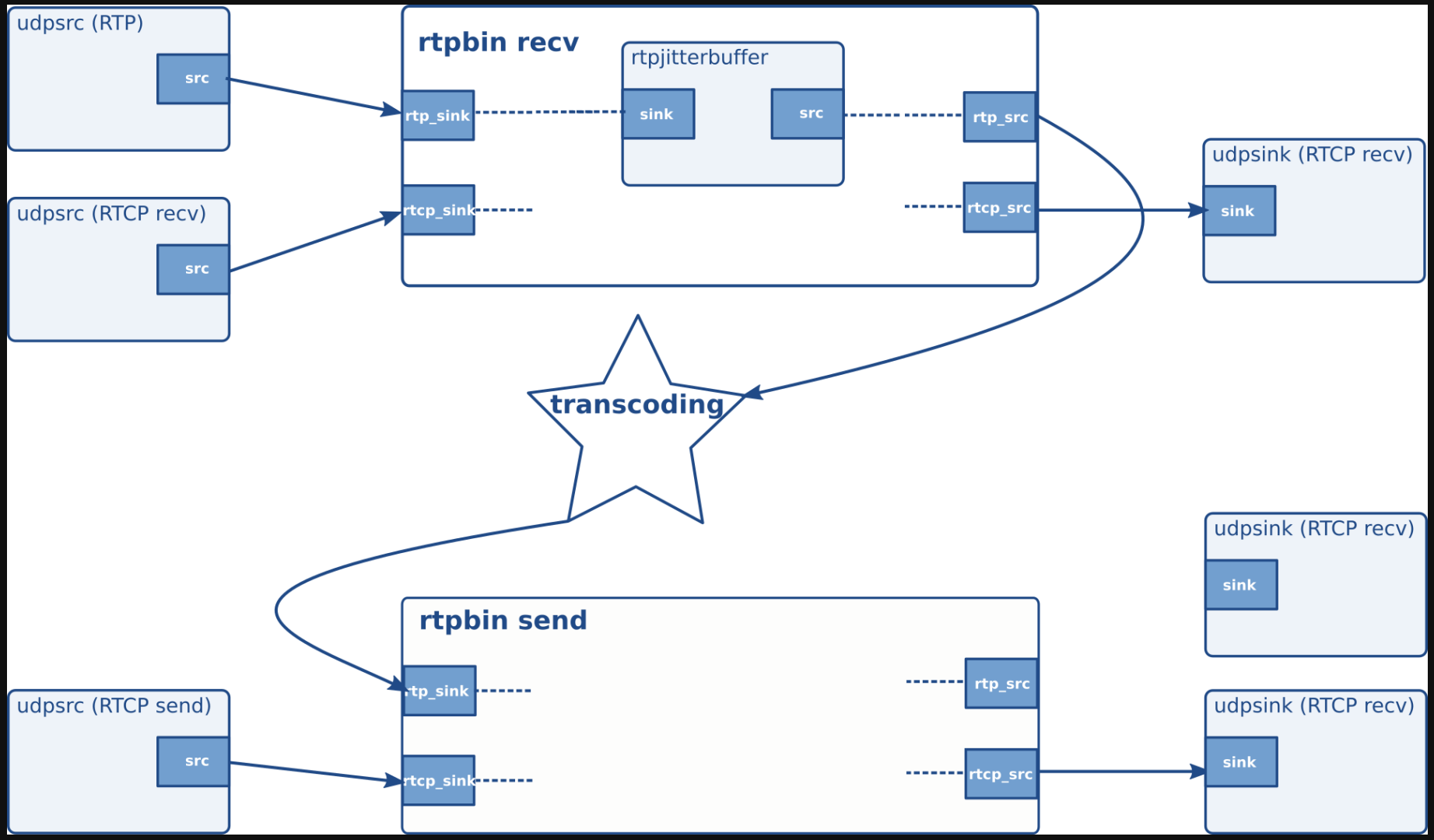
100x, 500x, 1000x in parallel on the same machine

Problem

7+ threads per pipeline

- Threads limit
- High CPU usage because of context switches, scheduler overhead
- Every thread doing nothing most of the time

Which threads do we have?


udpsrc (RTP)
src
udpsrc (RTCP recv)
src
rtpbin recv
rtp_sink
rtcp_sink
rtpjitterbuffer
sink
src
rtp_src
rtcp_src
udpsink (RTCP recv)
sink
transcoding
udpsink (RTCP recv)
sink
rtpbin send
rtp_sink
rtcp_sink
rtp_src
rtcp_src
udpsrc (RTCP send)
src
udpsink (RTCP recv)
sink

What can we do about this?

The Solution

Share threads between elements and pipelines

Goal: Use a fixed number of threads per process

RTP jitterbuffer

- Source pad thread: output independent of input
- Timer thread: Lost packets, RTX, ...

We don't need either of this!

New RTP jitterbuffer

Completely driven by packet receival

2 threads gone, 5 more to go

RTCP scheduling

- Timer thread: RTCP once every few seconds
- One for receiver, one for sender

This thread is sleeping most of the time!

New RTCP Manager element

- Can be set as manager on multiple rtpbins
- Receives and sends RTCP packets
- One timer thread

2x1 thread gone, 3 more to go

UDP sources (+ queues, etc.)

GstBaseSrc is creating one thread per source

By design: simplification!

Wouldn't it be nice to have as many threads as CPU cores and each of them polls a group of sockets?

gst-plugin-threadshare

- https://github.com/sdroege/gst-plugin-threadshare/
- Written in Rust, based on tokio.rs
- udpsrc, queue, proxysrc/sink, tonesrc, appsrc, tcpclientsrc, ...

Design: Context

- Named context shared by property name
- Context: 1 thread with poll() loop, throttling
- Register: sockets, timeouts, dispatch callbacks
- Poll thread dispatches callbacks

Design: Context Sharing

- Context is shared in pipeline branch (events)
- Allows all elements to register async operations

Design: Consumers, sinks

- Return OK when blocking
- Register callback: notify when ready
- Upstream waits async for downstream notify

- Implements backpressure

Design: Poll throttling

- Based on expected packet spacing, timer granularity
- Fewer wakeups, fewer context switches
- Handle all events in batches

3 more threads gone, 0 left!

Results

Fixed number of threads per process

1000+ pipelines on the same machine

Some Measurements

Element	Config	CPU
udpsrc	1000x	44%
ts-udpsrc	1000x / 1 context / 0ms	18%
ts-udpsrc	1000x / 1 context / 20ms	13%
ts-udpsrc	1000x / 2 context / 20ms	15%

Element	Config	CPU
udpsrc	2000x	95%
ts-udpsrc	2000x / 1 context / 20ms	29%
ts-udpsrc	2000x / 2 context / 20ms	31%

Element	Config	CPU
udpsrc	3000x	Failed
ts-udpsrc	3000x / 1 context / 20ms	36%
ts-udpsrc	3000x / 2 context / 20ms	47%

Conclusion

Future Work

Add a generic interface for thread-sharing

Make GStreamer optionally non-blocking for 2.0

Don't worry!

Should never be the default

Thanks! Questions?

sebastian@centricular.com

https://github.com/sdroege/gst-plugin-threadshare

https://coaxion.net/blog/2018/04/improving-gstreamer-performance-on-a-high-number-of-network-streams-by-sharing-threads-between-elements-with-rusts-tokio-crate/