# HIGH PACKET RATES IN GSTREAMER

GStreamer Conference

22 October 2017, Prague

Tim-Philipp Müller < tim@centricular.com >

## MORE OF A CASE STUDY REALLY

## HIGH PACKET RATES ?

## RTP RAW VIDEO: PACKET AND DATA RATES

- 1080p30 I420: ~ 68k packets/second @ mtu 1400
  97 MB/s ~= 0.7 Gbps
- 1080p30 UYVY: ~ 90k packets/second @ mtu 1400
  121 MB/s ~= 0.9 Gbps
- 1080p50 UYVY: ~150k packets/second @ mtu 1400
  201 MB/s ~= 1.6 Gbps
- 1080p50 RGB: ~225k packets/second @ mtu 1400
  302 MB/s ~= 2.3 Gbps

## RTP RAW VIDEO: PACKET AND DATA RATES

- 2160p30 I420: ~270k packets/second @ mtu 1400
  362 MB/s ~= 2.8 Gbps
- 2160p30 UYVY: ~360k packets/second @ mtu 1400
  482 MB/s ~= 3.8 Gbps
- 2160p30 I420: ~540k packets/second @ mtu 1400
  724 MB/s ~= 5.7 Gbps
- 2160p60 UYVY: ~720k packets/second @ mtu 1400
  964 MB/s ~= 7.5 Gbps

## SDI-OVER-IP / SMPTE 2022-6:

- 720p59.94: ~135k packets/second
  SDI bitrate ~= 1.5 Gbps
  one packet every 7.4 uSec
- 2160p50 @ 10-bit
  4:2:2, 10-bit 4:4:4, 12-bit 4:4:4 => 8, 12, 15 Gbps

## CHALLENGES

- capture
- processing
- sending

## GOALS

Optimise for throughput + low cpu usaage

## ANTI-GOALS

Optimise for lowest possible latency

## CAPTURE

At very high rates, it's a challenge to even capture packets fast enough.

Kernel bypass / zero-copy buffers

Lots of capture threads

recvmmsg(): multiple packets with one syscall

BUT WE'RE NOT GOING TO TALK ABOUT THIS,
WELL DESCRIBED ELSEWHERE.

## CAPTURE STATE OF THE ART IN GSTREAMER

Much more pedestrian, non-zero copy

Current udpsrc does one recvmsg() per packet, pushes out one buffer per packet received.

Bugzilla: use recvmmsg(), pushes out one buffer per packet

Tim's HD: use recvmmsg, push out a buffer list

## PROCESSING OVERHEAD

- passing buffers through the pipeline
- buffer allocation
- accessing buffer data
- processing buffer data

## BUFFER PASSING

Passing buffers has a cost.

Push tens of thousands or hundreds of thousands of buffers per second, and it will show up in your profiling.

## PAD PUSH INVOLVES:

Taking/releasing object locks multiple times.
(Cheap, atomic, uncontended)

Recursive lock (pad stream lock).
(Less cheap, at least on Linux, but not expensive).

Refs/unrefs, look-up peers etc. (Minor, but does add up!)

Add contention to the mix and it gets much worse!
(Queues, jitterbuffer, thread boundaries)

WE HAVE A SOLUTION FOR THIS IN GSTREAMER!

## GSTBUFFERLIST: WE CAN PASS N BUFFERS IN ONE GO! \O/

Reduces data passing overhead massively.

Is backwards compatible, transparently.

Just have to make sure every element in the pipeline maintains the buffer list for best perf.

## BUFFER ALLOCATION/FREEING

- single buffer: malloc/init/free for GstBuffer
- buffers need memories, so same for GstMemory
- *could* theoretically allocate buffer + memory chunk in one go, but not implemented yet
- can reuse buffers: buffer pools
- easier for capture (one buffer = one packet = one chunk/part of a chunk)

## BUFFER DATA ACCESS

- gst_buffer_map/unmap()
- gst_memory_map/unmap()
- gst_rtp_buffer_map/unmap()
- atomic ops, maybe even RTP header parsing
- could have a rtp/udp buffer pool that avoids mapping/unmapping
  the memory when filling (benefits unclear/unmeasured)

## GETS SILLIER WHEN PAYLOADING/SENDING

- GstBuffer can have multiple memories
- so we can prepend a GstMemory
- plus add a sub-memory (ALLOC) pointing into data of a parent GstMemory

OR

- alloc a new buffer, write RTP headers, memcpy payload data (MEMCPY)

**HEADS YOU LOSE. TAILS YOU LOSE.**

In order to process 50,000 buffers per second we involve
100,000-150,000 mini objects.

At least. Add some more for buffer lists and such.

Plus extra allocs for GstMetas.

Imagine how many times we map/unmap/parse even if we didn't have to allocate anything.

## BUFFER DATA PROCESSING

Let's look at the case of raw video processing.

Let's take packed UYVY or such. 1080p. 50fps.

Simples. One plane. Couple of MB per frame.

Now let's payload it into RTP packets @ MTU 1400

--> ~150k packets/second @ mtu 1400, 201 MB/s ~= 1.6 Gbps

What will the raw video payloader do? Lots of memcpy.

Still better than submems.

## MITIGATION STRATEGIES

Just use higher MTU. Might be possible in dedicated networks.

But often not.

## WHAT TO DO?

We want to avoid memcpy.

We want to re-use the raw video buffer.

Without allocating lots of things.

## PROPOSAL TIME!

We need to decouple packets from buffers and memories!

## ENTER GSTPACKETLIST

(Better name needed!)

## "GSTPACKETLIST"

Express N packets in terms of memory slices.

Can attach GstMemories to the packet list.

Each slice can refer to an attached GstMemory or "allocated header" slce.

We know how much space we'll need for headers and can pre-alloc a scratch area for RTP headers when we allocate the GstPacketList.

## RESULTS: 20X FASTER RTP PAYLOADING

First naive iteration.

Can be made even faster for rtp raw video by using templating for memory slices.

## OPEN ISSUE: GSTMETA

Could also be allocated inline, needs new APIs.

But not many metas are interesting here.

## OPEN ISSUE: ZERO-COPY TO UDPSINK

More work needed. No mechanism with kernel yet.

## WHERE'S THE CODE ?

Should land in bugzilla for discussion "soon".

Needs some cleaning up first.

QUESTIONS? COMMENTS?
OTHER IDEAS?