COLLABORA

Revisiting RTP Jitter Buffer Timers

by Nicolas Dufresne

IRC: ndufresne

Email: nicolas.dufresne@collabora.com

Open First

COLLABORA

Who am I ?

- GStreamer Developer
- Rock Climber
- A dad
- But mostly, I like to talk

COLLABORA

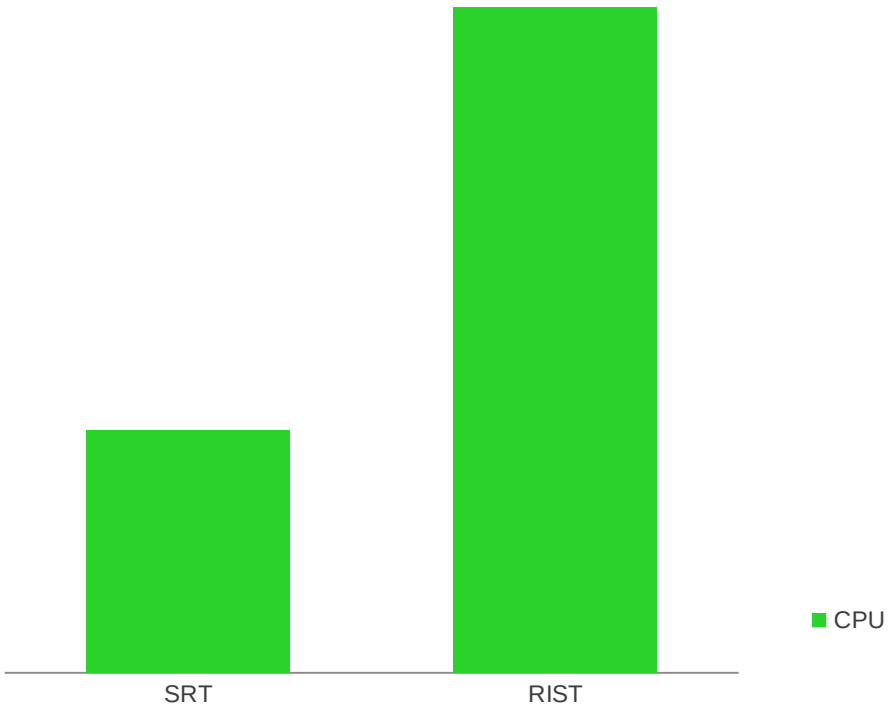
Once upon a time ...

While working on RIST

What is RIST ?

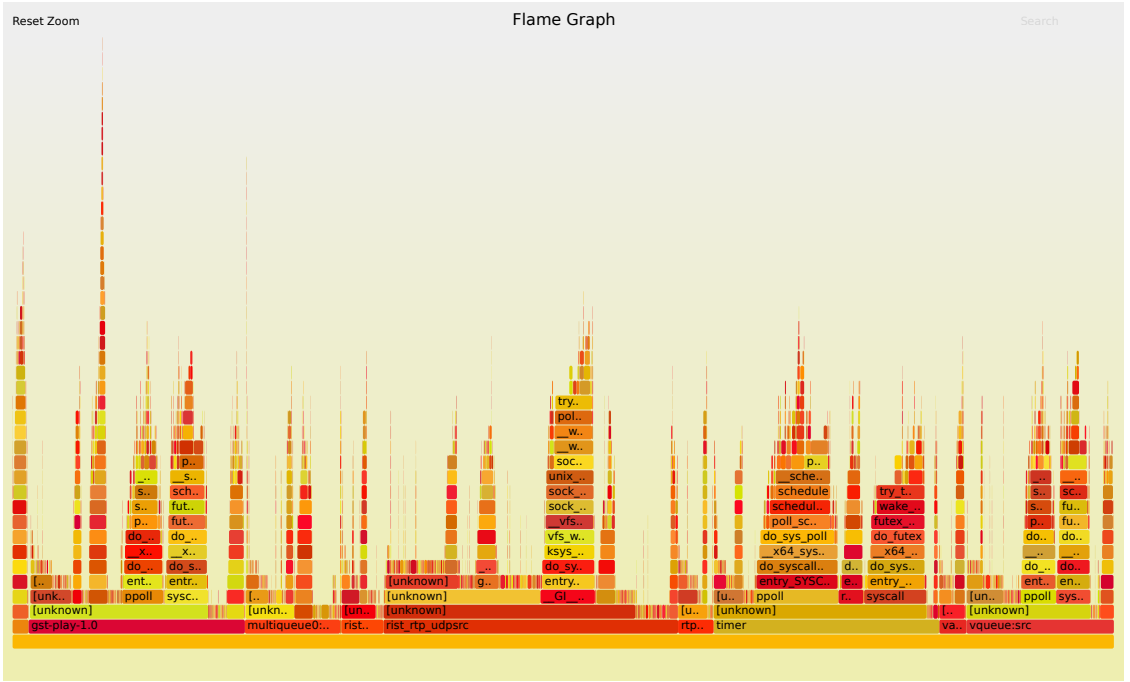
- RTP Based Protocol
- Transmitter / Receiver elements (sink and source)
- Does not require signaling
- Uses RTP Re-transmission
- Supports NACK Range
- Support Bonding (using multiple network links)
- But one problem remained …

RIST receiver used 3 times the CPU SRT uses



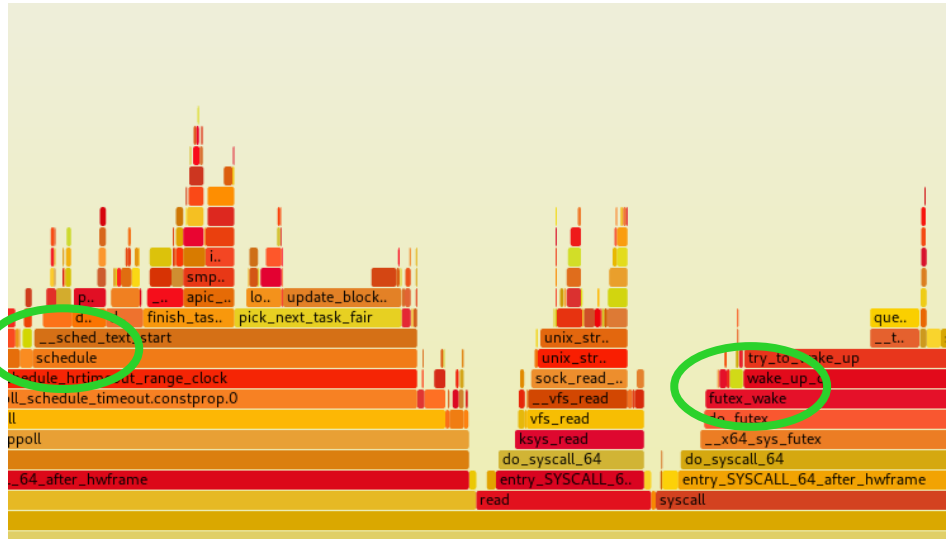
This made little sense since RIST is supposed to be simpler compared to SRT.

Flame Graph



http://www.brendangregg.com/flamegraphs.html

Spending time “scheduling” and “waking up”



Not a kernel expert, but scheduling work is offloaded to the process being preempted.

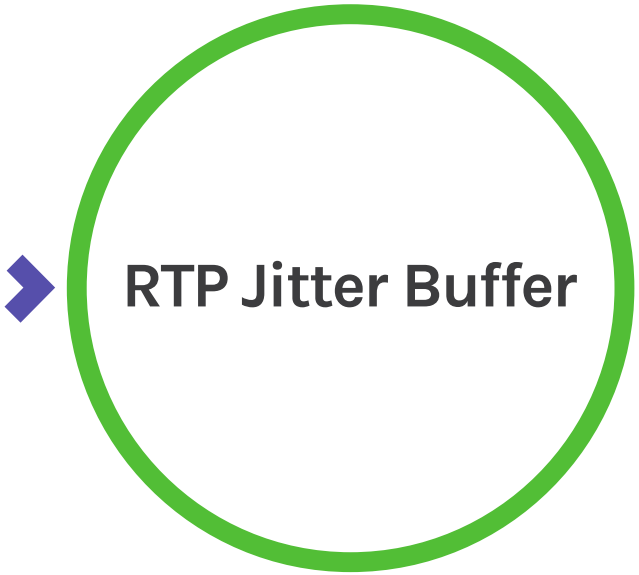
COLLABORA

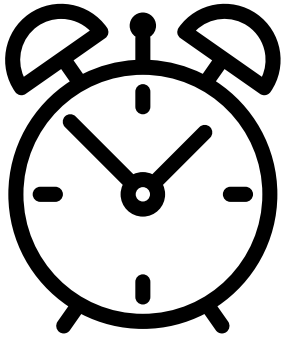
The Wake Up Theory

COLLABORA

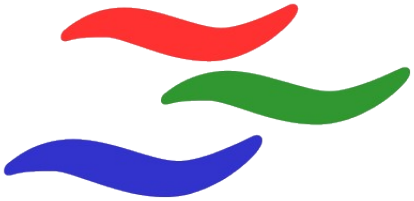
How to make sure that theory is correct ?

- Locate and read the code
- Understanding the threading model
- Find potential pathological interactions between threads



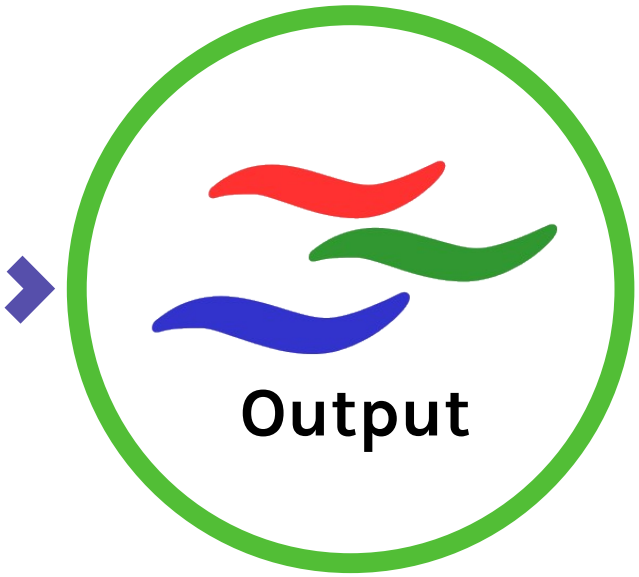


Timer

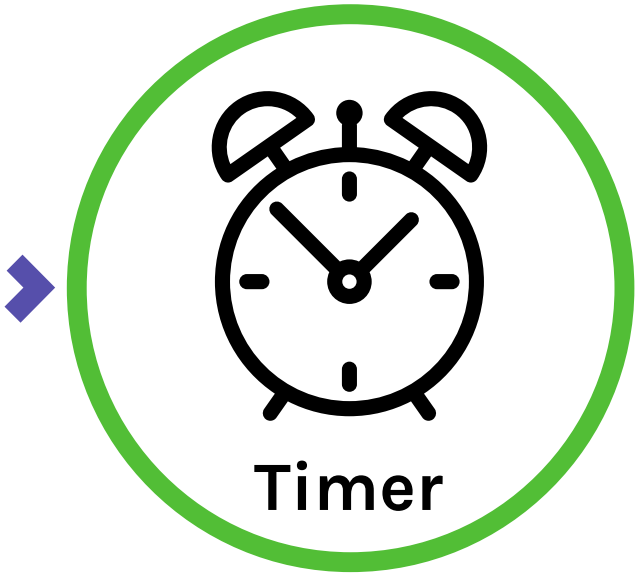


Input

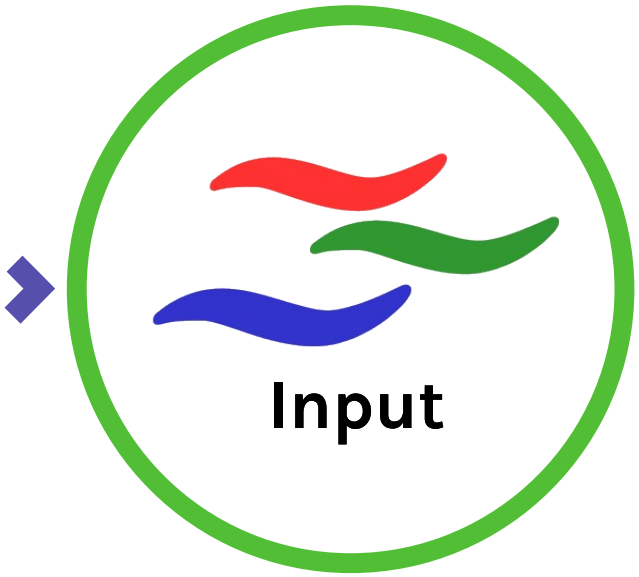
Output



- Pushes packets or gaps when they are ready
- Otherwise, waits on the jitterbuffer (an ordered list of packets)
- Is driven by data input and timers



- The thread executing timer specific actions
- Handles **two** sets of timers, an **unsorted array** of normal timers and an mostly ordered list of stats timers
- Can wake up the output thread on deadline and lost packets



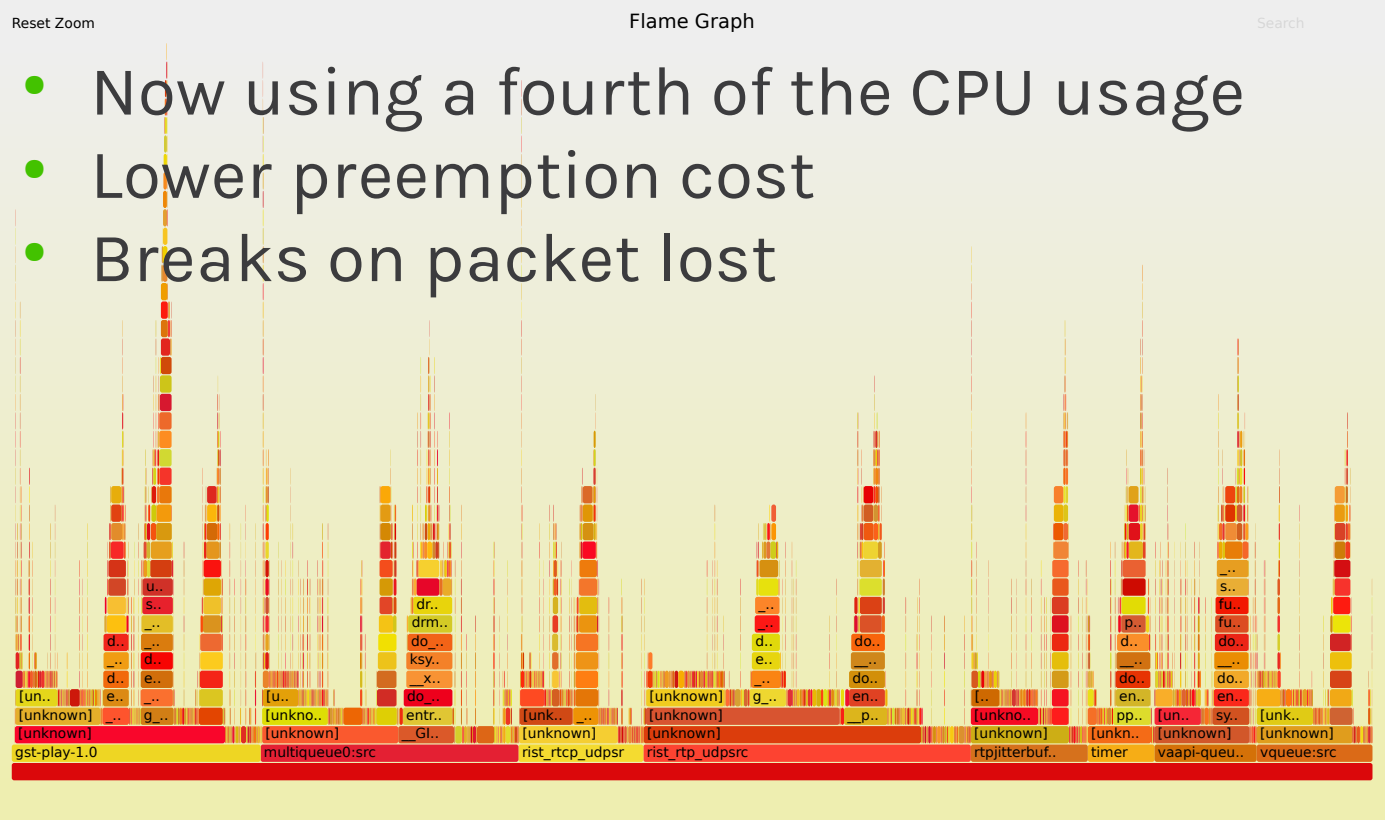
- Can wake up the timer thread **multiple** times per call when timers are **added**, **removed** or **rescheduled**
- Can also wake-up the output thread when data is in order
- **Does not know the next timer**, hence wake up the timer thread regardless of any logic

COLLABORA

The Proof of Concept

- Let’s never wake up the timer thread
- In normal condition, streaming should not be timer driven
- Commenting out *gst_clock_id_unscheduled ()*

The Proof of Concept



- Now using a fourth of the CPU usage
- Lower preemption cost
- Breaks on packet lost

COLLABORA

The Final Solution

There was at least two …

GstPriQueue

- Presented in 2017 by Erlend Graff
- Focused on data structure speed
- A bit complicated

Stats Timer Queue

- Optimized O(1) lookup
- O(1) access to next timer
- Simple to understand

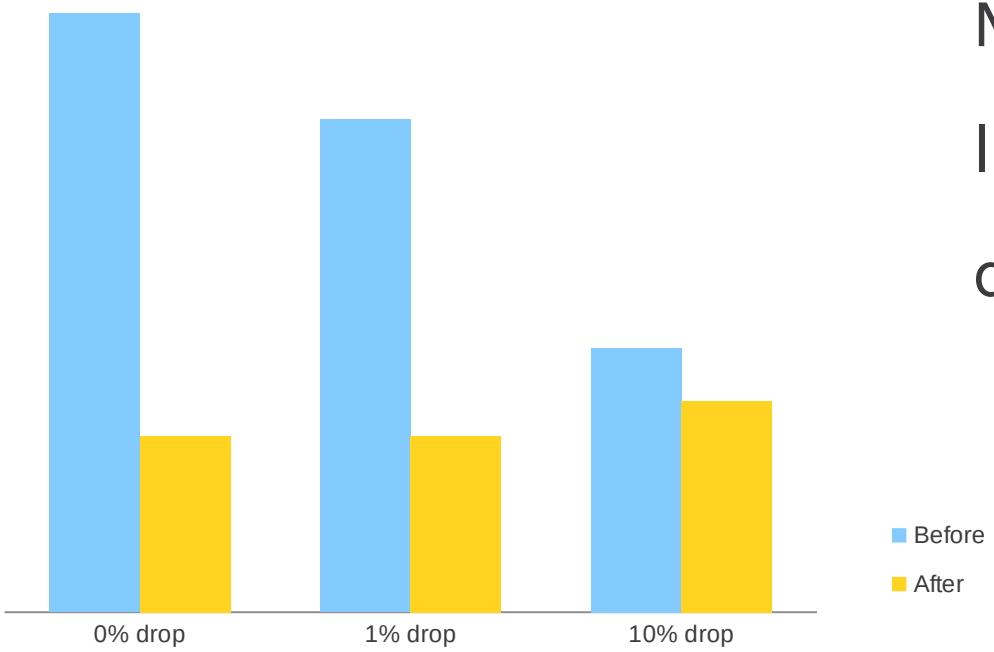
Reusing existing queue ?

- Kept the hash table O(1) packet lookup
- But embedded GList into RtpTimer structure, saving loads of allocations
- Implemented sorted insertions and rescheduling
- Added neighboring lookup to speed up rescheduling
- Split it out of the GST code, rtptimerqueue.h/c
- Added a good set of unit test for it

COLLABORA

Two weeks or so later …

It started to pay off



Massive speed up when no lost, and small gain on 10% data lost.

COLLABORA

Will land in 1.18

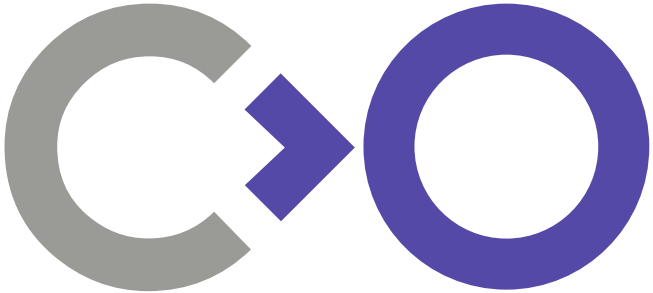
- Makes the rtpjitterbuffer timers easier to work with
- Trying out GstPriQueue should be straightforward

COLLABORA

What's left ?

- Keep fixing unit tests that relied on the old behaviour
- Gather feedback from other users, fix any regression

Question ?



Thank you!