What makes Upipe great for video processing
What is Upipe?

- A young (2012) C multimedia framework
- Initiated by OpenHeadend team
- 3 supporting companies, 7 contributors
- Pre-release 4 just out, plans for 1.0 to be discussed
- Focus on reliability, efficiency and compliance, for broadcast and professional applications
- MIT and LGPL
Upipe main structures

- `struct upipe`: unit of data processing, part of a pipeline
- `struct uprobe`: internal exception catcher, interaction with application
- `struct upump`: external event catcher, event loop
- `struct uclock`: system clock management
- `struct urequest`: inter-pipe negotiation
- `struct uref`: unit of data and metadata
- `struct ubuf`: container of data
What makes Upipe great for video processing
Advanced buffer management

- Refcounts allow buffer-sharing
  - `struct ubuf *new = ubuf_dup(ubuf);`
- Copy-on-write semantics
  - `ubuf_block_write(ubuf, offset, &size, &buf);`
  - `new = ubuf_block_copy(ubuf_mgr, ubuf, skip, new_size);`
- Zero-copy primitives for data blocks
  - `ubuf_block_append(ubuf1, ubuf2);`
  - `ubuf_block_insert(ubuf1, offset, ubuf2);`
  - `ubuf_block_delete(ubuf, offset, size);`
  - `ubuf_block_resize(ubuf, offset, new_size);`
Arbitrary meta-data attributes

- struct uref carried across pipes embeds timestamps & pointers to ubuf and attribute dictionary
- Preprocessor macros allow easy attribute declarations:
  - UREF_ATTR_UNSIGNED(foo, bar, "foo.bar", example of attribute)
- Accessors to manipulate the attribute:
  - uref_foo_set_bar(uref, 42);
  - uref_foo_get_bar(uref, &uint64_var);
  - uref_goo_delete_bar(uref);
- Implemented with inline buffer (no memory allocation)
Three clock variants

- Packet timestamps are stored in three clock variants:
  - **orig**: original timestamp scaled to 27MHz units
  - **prog**: same scale as **orig** but origin moved to be monotonically increasing, used to encode PTS/DTS
  - **sys**: scale of the system clock (**uclock**), used to output packets
- Allows to keep a compliant difference between PTSs based on frame rate (e.g. 40 ms)
- At the same time skew the system clock to output packets faster or slower to keep up with transmitter
Inner pipes

- Pipes can “embed” other pipes
- Allows for more granularity and flexibility
Event loop

- Upipe’s event loop can be exposed to all pipe types via an abstract API (upump)
- Allows to have timers in any pipe, for instance event handling (SCTE-35 splicing)
  - upump = upump_alloc_timer(upump_mgr, callback, opaque, refcount, timeout, repeated);
- Or to interact with the external world
  - upump = upump_alloc_fd_read(upump_mgr, callback, opaque, refcount, file_descriptor);
Dynamic pipeline construction

• The uprobe API allows the application to receive exceptions from the pipe.
• Application can be notified when a pipe needs an output (UPROBE_NEED_OUTPUT), or the list of elementary streams of a demux changes (UPROBE_SPLIT_UPDATE).
• Useful for dynamic formats such as transport streams.
Threading & buffering

• Pipes are low-level — threading is decided by the application
• Queues and workers can move pipelines to threads
  – local_pipe = upipe_wlin_alloc(remote_thread, local_probe, remote_pipe, remote_probe, input_queue_len, output_queue_len);
• Local pipe takes the place of the remote pipe in the local thread and can be acted upon
• Lockless FIFOs and LIFOs handle and recycle data structures + eventfd abstraction
Inter-pipe negotiation

- Pipes can register `struct urequest` on their output.
- The request is passed from pipe to pipe until handled, the reply flows backwards from callback to callback.
- Requests cross threads boundaries via queues and are resent in case of pipeline changes.
Available pipes

- **upipe-modules**: pipes for basic manipulation and I/O
- **upipe-ts**: standards-compliant TS demux and mux
- **upipe-framers**: bitstream conversion of common codecs (mp2v, h264, mp2, aac, a52, opus, telx, dvbsub, s302)
- **upipe-av**: avformat demux and mux, avcodec decoder and encoder
- **upipe-swresample, upipe-swscale, upipe-x264, upipe-blackmagic**
Application development

• Currently C API
• Work for LuaJIT bindings under way: https://github.com/nto/lj-upipe
• Transcode example application:
  
  Usage: transcode [-d] [-m <mime>] [-f <format>]
            [-p <id> -c <codec> [-o <option=value>] ...] ...
            <source file> <sink file>
  -f: output format name
  -m: output mime type
  -p: add stream with id
  -c: stream encoder
  -o: encoder option (key=value)
What makes Upipe great for video processing

cmassiot@openheadend.tv
Upipe meet-up in BOF room Sunday 14:00