AMT use case: Upipe + Chrome

Christophe Massiot (EBU multicast 2014)
Goal

- Display a multicast stream in a web browser, using AMT if needed
- Without AMT support from the OS, or from a local network equipment
Case 1: Direct access to multicast

Multicast sender

Multicast-enabled network

UDP/RTP

IGMP

Multicast receiver
Case 2: Use AMT relay

- Multicast sender
- Multicast-enabled network
- IGMP
- UDP/RTP
- AMT Relay
- IP Network without multicast
- Multicast receiver
- AMT
How?

AMT use case: Upipe + Chrome
Open source AMT libraries

• Only one: Cisco’s
  – Written in C
  – Simple API: open, poll, recv
Display video in a web browser

• The old-fashioned way: NPAPI “Netscape plug-ins”
  – Blatant security hole (no sandbox)
  – Requires user installation
  – Mozilla & Google have announced end of support in 2014
Display video in a web browser

• Media Source Extensions
  – Designed for HLS/HDS/DASH etc.
  – Would have to implement AMT in JS
  – Would need UDP socket support from browser (only supported by Chrome, and for packaged applications)
  – Not as open as “native” extensions
Display video in a web browser

• Microsoft ActiveX
  – Example: VLC plug-in
  – No support for Mac OS X, Linux or embedded devices
  – Security issues
Display video in a web browser

- Google’s “PPAPI” and “native client”
  - Open source
  - Supports downloadable native code (C and C++) and dynamic linking
  - Plug-in isolated in a “sandbox”
  - Supported in recent Chrome versions on all operating systems, and plans to port to Android devices
  - Also has support for “portable” bitcode
Knitting the pieces together

• Need a C or C++ framework with demux/decode capabilities
• Must be very flexible: PPAPI has constraints on threading and API calls
Introducing Upipe

• Handles flows of data in a « pipeline »
• Processes them using filters called « pipes »
• Defines APIs:
  – To configure and feed data into pipes
  – To get out-of-band events from pipes
  – To store data in an efficient manner with attributes
  – To interact with an event loop/threads
• Provides a set of basic pipes
Upipe vs. other frameworks

• Flexibility
  – Lower-level: application decides where to put queues and threads
  – Keep modules as simple and autonomous as possible (UNIX philosophy)
  – Arbitrary attributes on buffers ➔ better clock management
  – Dynamic pipeline construction
Upipe vs. other frameworks

• Event-driven architecture (upump)
  – Pipes install watchers called back when events occur:
    • File descriptors readable or writable
    • Timers
    • Idlers
  – Non-blocking
  – Unified API for sources/filters/sinks
Upipe vs. other frameworks

• Performance
  – Lock-less or wait-less data structures
  – Systematic reference counts on structures
  – Zerocopy, copy-on-write buffer management

• Younger project: fewer modules, fewer users
Which steps?
Implementing AMT support

• Source file: lib/upipe-amt/upipe_amt_source.c
• Caveats: libamtm not compatible with event-driven architectures, so requires its own thread
A simple AMT-enabled player

• Upipe’s example uplay requires a 2-line change to use AMT source
Implementing NaCl support

- BSD sockets are supported with compatibility layer libnacl_io
- Need specific modules for display and sound (lib/upipe-nacl)
- Need a “player” plug-in building the pipeline, based on uplay (examples/chrome/player_chrome)
- Messaging between JS and plug-in
NaCl configuration

• NaCl and pNaCl enabled by default in recent Chrome versions
• Sockets blocked by default
  – chrome://flags only works for “packaged apps”
  – Need to pass command-line option
    --allow-nacl-socket-api=localhost
NaCl socket support

• Only a subset of BSD sockets API
  – Missing setsockopt options:
    • IP_ADD_MEMBERSHIP / DROP
    • IP_ADD_SOURCE_MEMBERSHIP / DROP
    • IP_PKTINFO
    • ...
  – Cannot bind to a multicast address

• AMT provides a work-around
NaCl performance

• “Nearly as fast as native compiled code” but in reality:
  – No assembly allowed (no decoder optimizations)
  – No easy YUV support

• Portable Native Client adds 10 to 25% overhead

• However, support for OpenGL and hardware decoders under way
Hands on!

AMIT use case: Upipe + Chrome
Installing NaCl SDK and NaCl ports


- Need build environment and 32 bits version of standard libraries, in particular lib32z1-dev
Building NaCl ports

• Apply
  http://upipe.org/downloads/naclports.patch

• export NACL_SDK_ROOT=<path>/nacl_sdk/pepper_35

• NACL_ARCH=x86_64 TOOLCHAIN=glibc ./bin/naclports install upipe

• NACL_ARCH=i686 TOOLCHAIN=glibc ./bin/naclports install upipe
Building and running plug-in

- cd out/build/upipe/upipe-master/examples/chrome/player_chrome
- make
- export CHROME_ARGS="--allow-nacl-socket-api=192.168.0.100,localhost"
- make run
AMT use case: Upipe + Chrome

cmassiot@openheadend.tv