



Adding ASR to an Akraino Robot

Signallogic, Inc.
Dallas, Texas

Contents

- **Executive Summary**
- **Problems**
 - cloud computing is not edge computing
 - long “supply chain” from cloud to edge
- **Robot Needs**
 - deal with environment at the edge
 - operate safely – don’t hurt anyone, don’t damage property. Operate with zero trust in cloud commands
 - operate with energy and size constraints
- **EdgeStream™ - Software**
 - optimized on per thread basis, one thread per core, no spinlocks
 - packet processing, media codecs, signal processing, inference
 - comparison with DeepStream and GStreamer
- **EdgeStream™ - Hardware**
 - pico ITX form factor: 3.5” x 3.5”
 - quad-core Atom, no fan
- **Deployments**
- **Github and Docker Hub**
 - Demos, reference apps, ready-to-run containers, example command lines
 - Source code

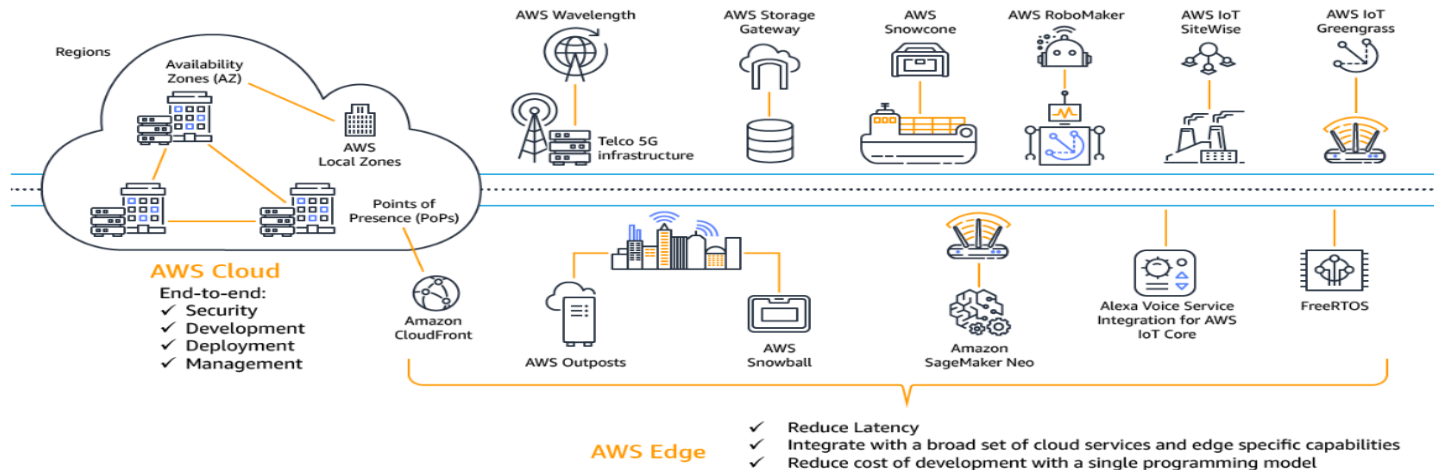
Executive Summary



- **Signallogic is adding ASR (automatic speech recognition) to an Akraino blueprint**
 - key blueprint organizations: Fujitsu and Ritsumeikan University
- **Robot compute resources are typically not sufficient for real-time, high accuracy, high vocabulary ASR**
- **Initial demo is a Roomba**
 - quad-core pico ITX board and battery “dead bugged” on a Roomba
 - voice commands are processed and sent to the Roomba’s API via USB; e.g. “stop”, “come back later”, “turn left”, etc.

Problems Facing Robotics ASR

- **Robots are a quintessential edge application**
 - compute resources and power consumption are strictly limited. There is no “economy of scale”
 - operation cannot depend on continuous internet connectivity
- **Human safety is paramount**
 - accuracy must be high – when someone says “stop, back up” robots must immediately do exactly that
- **Edge data is private data**
 - Images and audio may contain identifying / proprietary information. **Audio data cannot be sent to the cloud** over a “long supply chain” of zones, regions, services

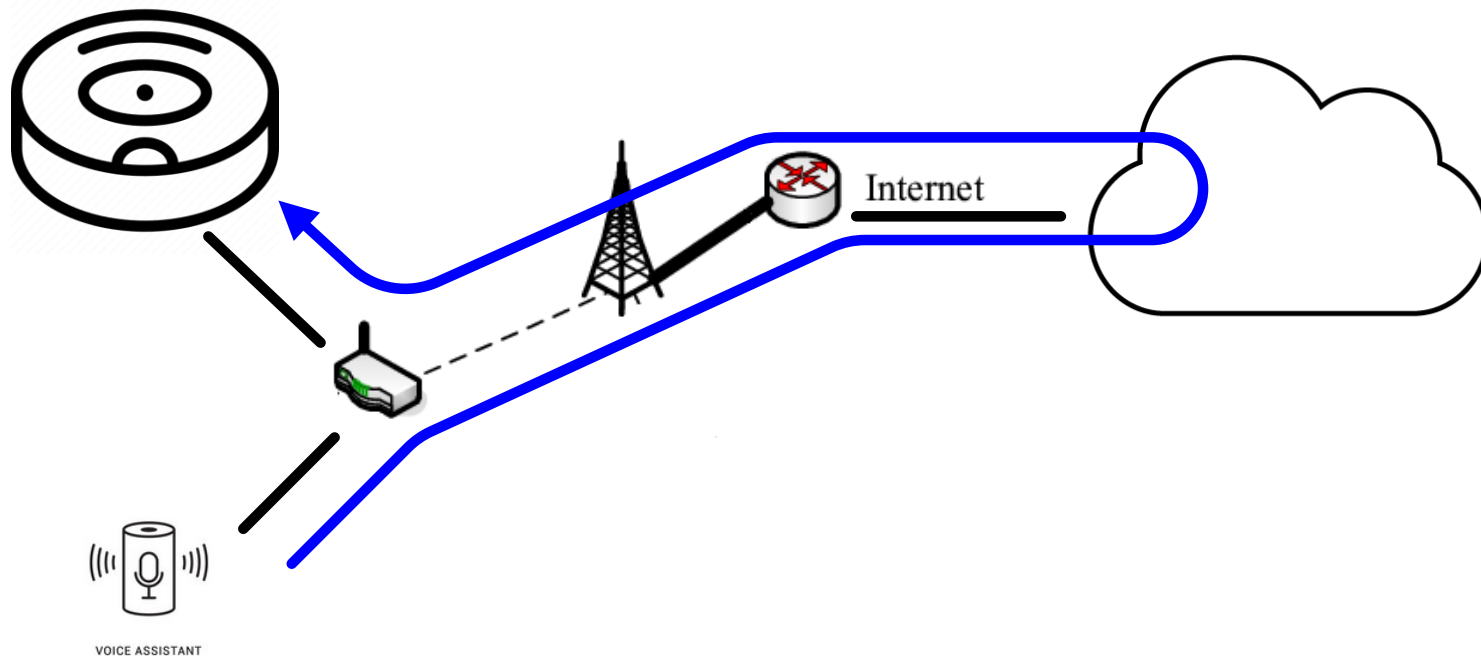


Source: AWS, 2020

Problems, example

- **Why talk to your roomba through the cloud ?**

- the “centralized cloud” model is based on huge scale, data collection, analytics, command and control, and complexity
- not based on energy and performance efficiency, safety, and data privacy



Robot Needs

- **Deal with environment at the edge**
 - respond to voice commands
 - vision - avoid people, pets, property. Don't break things
- **Operate safely - don't hurt anyone, don't damage property**
 - operate with zero trust in cloud commands – the measurable physical situation always takes precedence
 - actively pursue and detect malware
- **Share content with other edge nodes**
 - share private edge data – which will become massive amounts as we go forward
- **Energy and size constraints**
 - operate with 50 – 250 W power consumption, strive for “no fans”
 - operate in small-form factor boxes: 1/2 1U, mini-ITX, smaller



“Beat it roomba, not now”

Software – EdgeStream™ Platform



- **Input from USB audio or RTP packets**
 - input from one more microphones
- **Pre-processing (e.g. background noise removal)**
 - environment background noise
 - robot noise, e.g. motors, wheels, brushes
- **Application specific processing**
 - lawful intelligence / interception
 - telecom
- **ASR**
- **Output commands to connected devices as needed**
 - translate recognized text to command APIs

EdgeStream Data Flow

• Per core data flow

- one thread per core
- no spinlocks
- precise control over power consumption

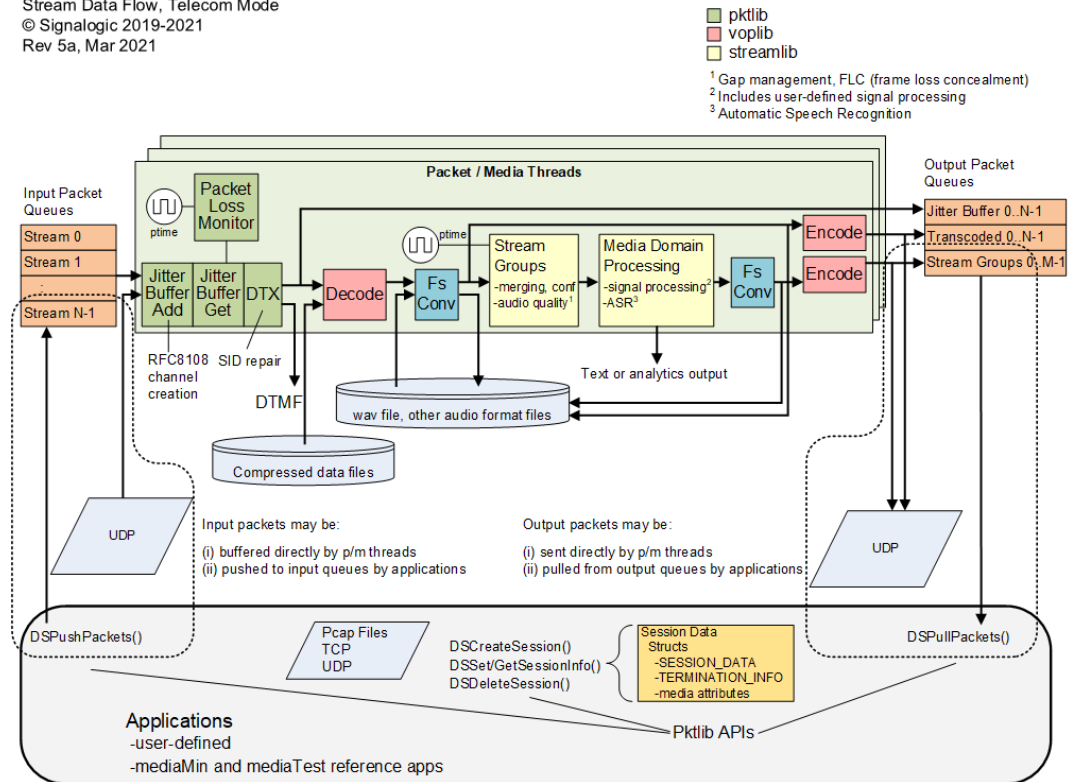
• Real-time workflow

- packet handling
- media codecs
- signal processing
- user-defined processing
- inference

• Hardware acceleration

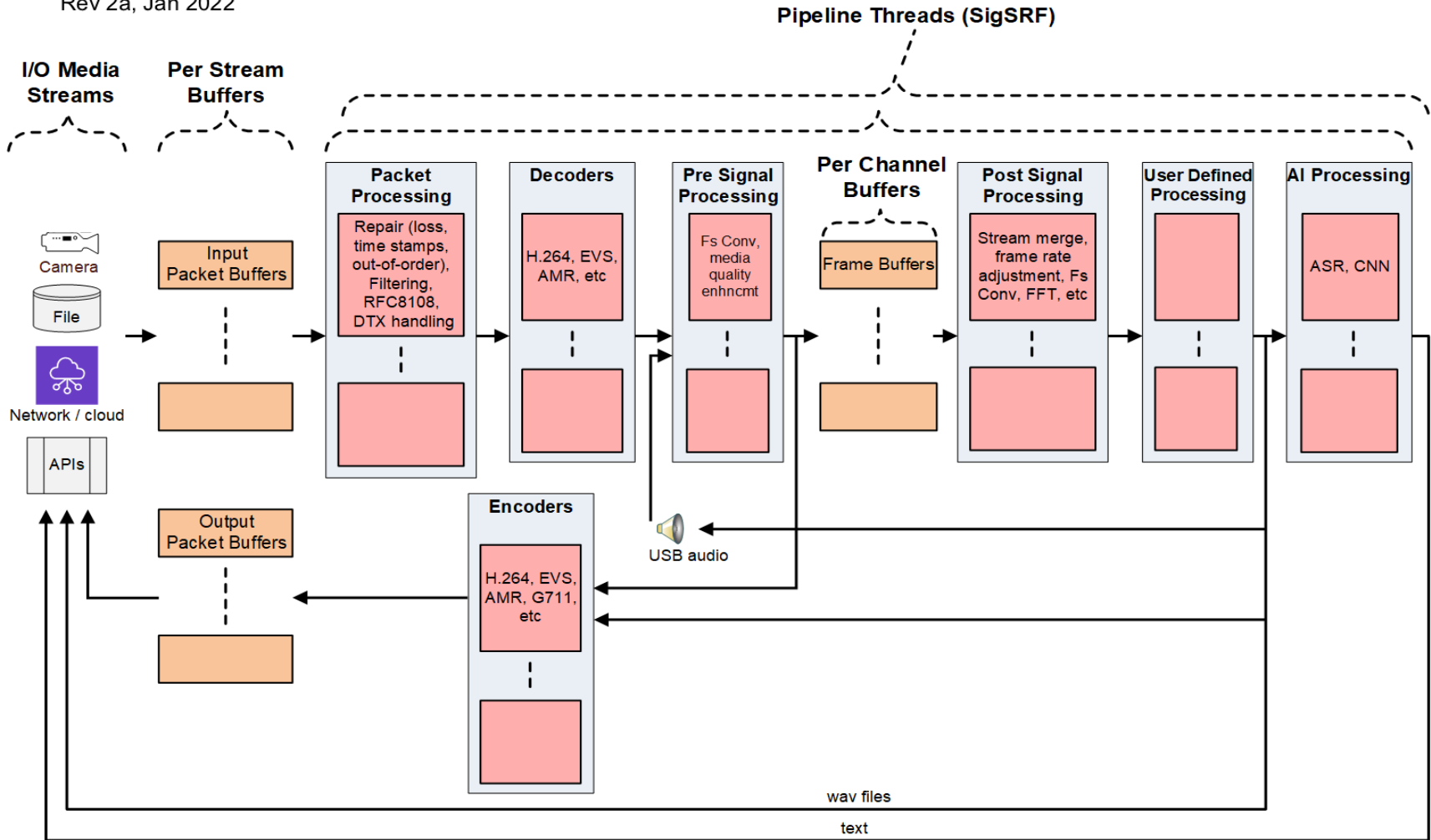
- DirectCore® option
- x86 and Arm options supported

Stream Data Flow, Telecom Mode
© Signallogic 2019-2021
Rev 5a, Mar 2021



EdgeStream Workflow

EdgeStream™ Workflow
© Signallogic 2021-2022
Rev 2a, Jan 2022



Comparison with DeepStream

- **Packet Processing**

- EdgeStream provides telecom grade packet processing, including...
- loss repair
- 500+ out-of-order handling
- support for encapsulated protocols
- multiple RFCs
- logging

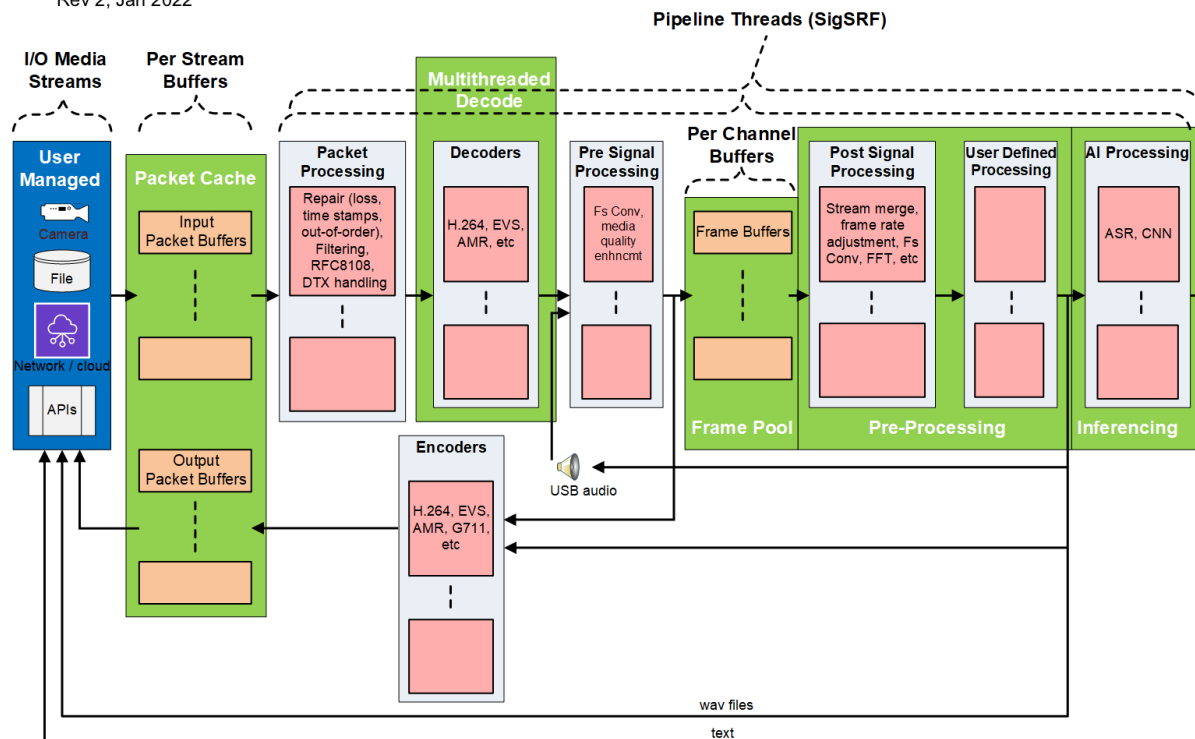
- **Media**

- includes encoders in addition to decoders

- **Signal processing**

- more user-defined insertion points

EdgeStream™ DeepStream Functional Mapping
© Signallogic 2021-2022
Rev 2, Jan 2022

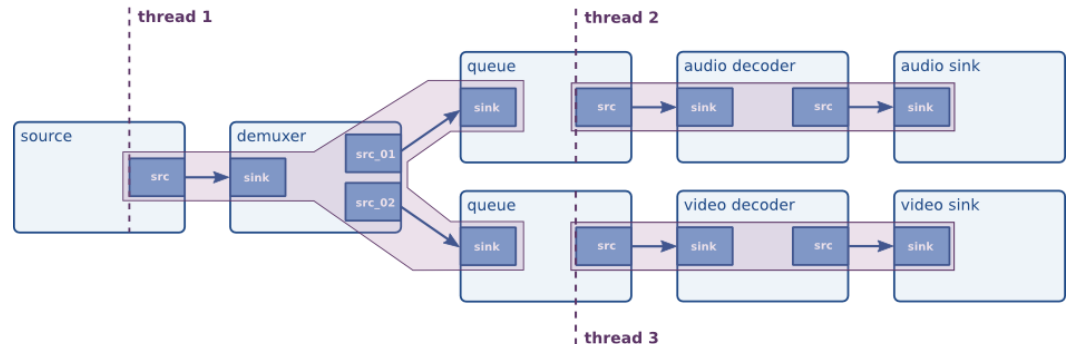


Comparison with GStreamer

• Thread architecture

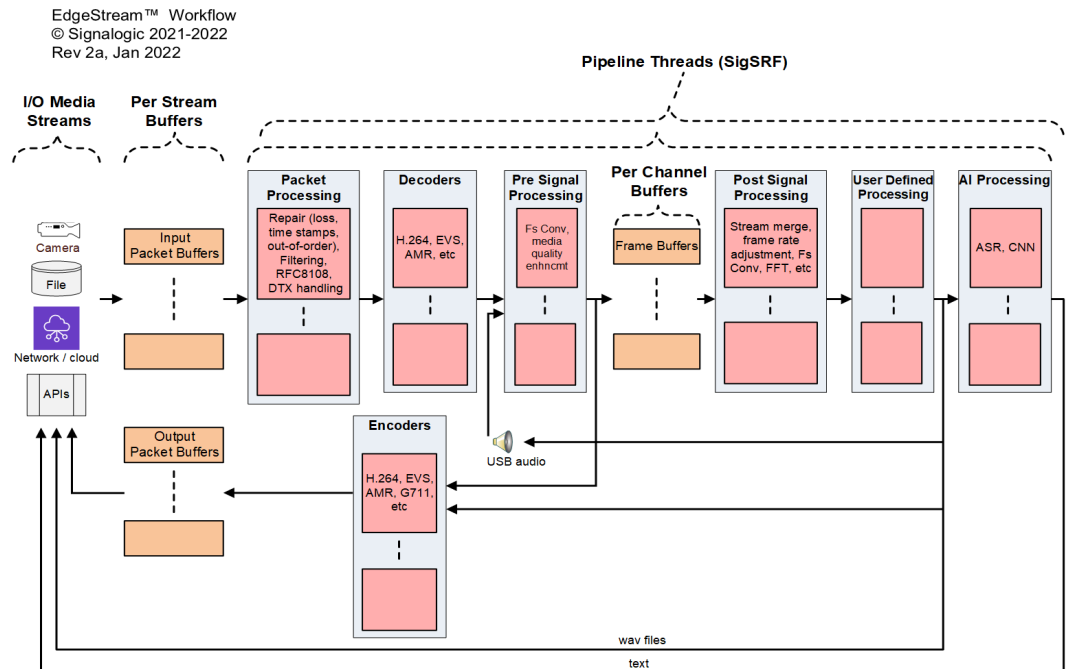
- EdgeStream allocates one workflow thread per core (“unified thread”)
- GStreamer uses a thread slicing architecture – flexible but requires spinlocks

Example GStreamer Workflow



• Packet Processing

- EdgeStream provides telecom grade packet processing, including ...
- loss repair
- 500+ out-of-order handling
- supports encapsulated protocols
- multiple RFCs
- logging



Hardware – EdgeStream™ Platform

- **Pico ITX form-factor board**
 - 3.5" x 3.5" form-factor
 - quad-core Atom, 10 to 20 W power consumption
 - integrated SSD, WiFi, HDMI, etc.
 - no fan
 - boots Ubuntu 20.04
- **Essentially a “robot server”**
 - straightforward to develop code on lab and cloud servers, then run on a robot server
 - lab / cloud servers can easily simulate robot servers by controlling number of cores and clock rate
 - handles up to three (3) concurrent ASR streams, or three (3) far-field microphones for one stream, or a combination



EdgeStream Deployments

- **Asia**

- Japan
- India (ISRO)
- Australia
- New Zealand (OpenLI ¹ support)

- **Europe**

- Germany
- Italy
- Czech Republic

- **North America**

- AFRL
- Raytheon
- Boeing

¹ OpenLI is “Open Lawful Intercept” for CSPs. More info at <https://openli.nz/>

Github and Docker Hub

- **Github**

- SigSRF software page: https://github.com/signallogic/SigSRF_SDK
- example command lines for reference apps and demos
- documentation

- **Docker Hub**

- ready-to-run Ubuntu and CentOS containers <https://hub.docker.com/u/signallogic>

- **Demos and reference apps**

- ready-to-run containers on Docker Hub, installation Rar packages on Github
- help with installing and running demos available over Skype (no charge)

- **Source code**

- developed entirely in US
- no dependencies on 3rd party libraries

Thanks !



- Q&A
- Follow-up questions / comments: info@signallogic.com
- Web page: <https://signallogic.com/edgestream>

Supplemental

- **Following slides are background info ...**

Overview

• SigSRF libraries

– codecs

- VoLTE (EVS, AMR-NB, AMR-WB)
- legacy (G729, EVRC, GSM, etc)
- mil/gov (MELPe)

– packet processing

- media/SID packet repair (out-of-order, packet loss, RTP timestamps)
- timing reconstruction of missing/damaged arrival timestamps
- child streams (RFC8108)

– frame processing

- “stream groups” can be defined for related streams
- per-stream correction for overrun, underrun, gaps, bursts
- accurate time-aligned merging / mixing of multiple endpoints
- high capacity – multiple concurrent streams

• EdgeStream™ applications

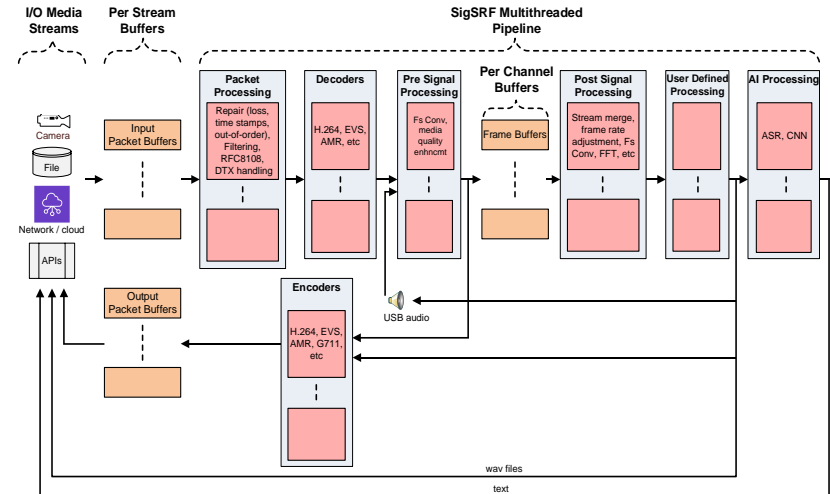
– reference apps for customer-defined development

– also used as-is by many of our customers. Most common: telecom, LI, and ASR.

– key features

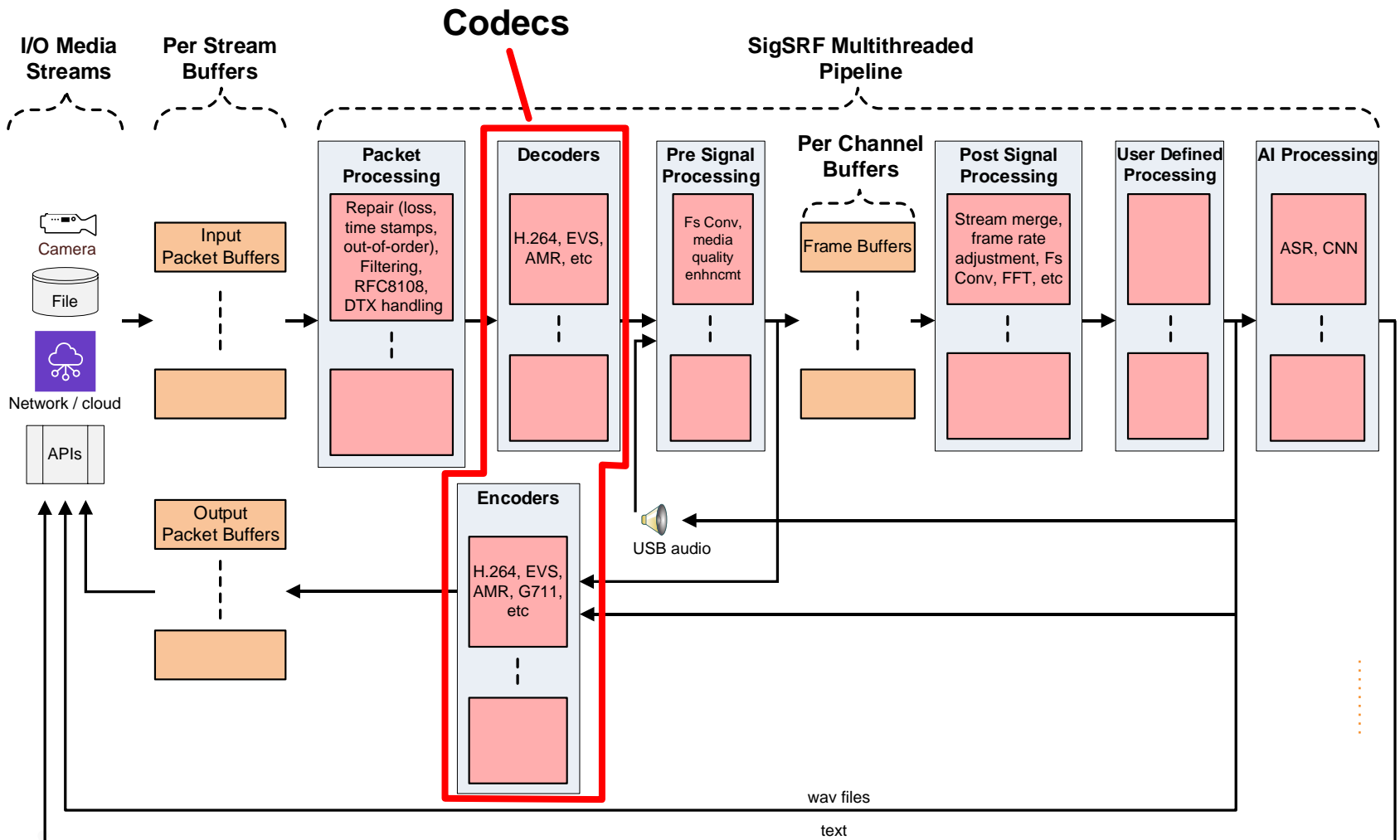
- dynamic session creation
- packet push/pull API interface with SigSRF libs
- multiple streams from multiple sources
- flexible command line – similar to ffmpeg or sox

EdgeStream™ Workflow
© Signallogic 2021
Rev 1a, Dec 2021



Overview – Pipeline Flow

EdgeStream™ Workflow
© Signallogic 2021
Rev 1a, Dec 2021



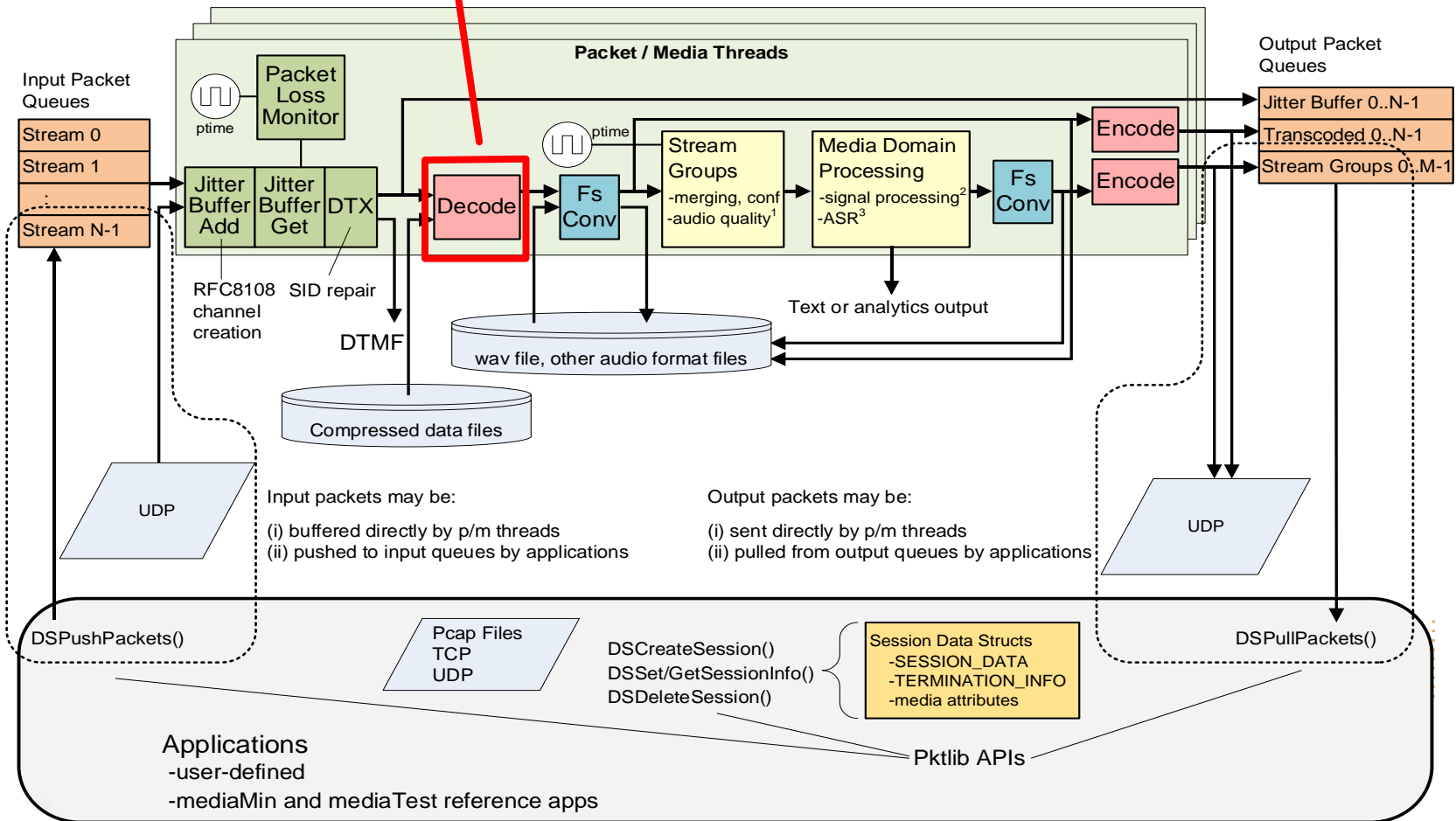
Overview – Per Thread Data Flow

Stream Data Flow, Telecom Mode
 © Signallogic 2019-2021
 Rev 5a, Mar 2021

■ pktlib
■ voplib
■ streamlib

¹ Gap management, FLC (frame loss concealment)
² Includes user-defined signal processing
³ Automatic Speech Recognition

Codecs



Functionality – Packet Processing

- **Media quality – packet level**

- media/SID packet repair
 - out-of-order (ooo)
 - packet loss
 - RTP timestamps
- child streams (RFC8108)
- timing reconstruction for missing/damaged packet arrival timestamps

- **Huge levels of ooo handled**

- to support TCP encapsulated UDP/RTP, for example lawful interception apps implementing ETSI protocols

- **Packet logging / tracing**

- per stream packet logging
- timestamp reconciliation
- individual packet tracing

Ingress Packet info for SSRC = 0xbad52e64, first seq num = 3, last seq num = 651 ...

Seq num 4 ooo 3	timestamp = 1280, pkt len = 6 SID
Seq num 3 ooo 4	timestamp = 960, pkt len = 61
Seq num 5	timestamp = 3840, pkt len = 6 SID
Seq num 6	timestamp = 6400, pkt len = 6 SID
Seq num 7	timestamp = 8960, pkt len = 6 SID
Seq num 8	timestamp = 11520, pkt len = 6 SID
Seq num 9	timestamp = 14080, pkt len = 6 SID
Seq num 10	timestamp = 16640, pkt len = 6 SID
Seq num 12 ooo 11	timestamp = 18560, pkt len = 61
Seq num 15 ooo 12	timestamp = 19520, pkt len = 61
Seq num 11 ooo 13	timestamp = 18240, pkt len = 61
Seq num 13 ooo 14	timestamp = 18880, pkt len = 61
Seq num 14 ooo 15	timestamp = 19200, pkt len = 61
Seq num 18 ooo 16	timestamp = 20480, pkt len = 61
Seq num 19 ooo 17	timestamp = 20800, pkt len = 61
Seq num 16 ooo 18	timestamp = 19840, pkt len = 61
Seq num 21 ooo 19	timestamp = 21440, pkt len = 6 SID
Seq num 23 ooo 20	timestamp = 23680, pkt len = 61
Seq num 24 ooo 21	timestamp = 24000, pkt len = 61
Seq num 25 ooo 22	timestamp = 24320, pkt len = 61
Seq num 27 ooo 23	timestamp = 24960, pkt len = 61
Seq num 28 ooo 24	timestamp = 25280, pkt len = 61
Seq num 31 ooo 25	timestamp = 26240, pkt len = 61
Seq num 32 ooo 26	timestamp = 26560, pkt len = 61
Seq num 34 ooo 27	timestamp = 27200, pkt len = 61
Seq num 17 ooo 28	timestamp = 20160, pkt len = 61
Seq num 20 ooo 29	timestamp = 21120, pkt len = 61
Seq num 22 ooo 30	timestamp = 23360, pkt len = 61
Seq num 26 ooo 31	timestamp = 24640, pkt len = 61
Seq num 29 ooo 32	timestamp = 25600, pkt len = 61
Seq num 30 ooo 33	timestamp = 25920, pkt len = 61
Seq num 33 ooo 34	timestamp = 26880, pkt len = 61
Seq num 35	timestamp = 27520, pkt len = 61
Seq num 37 ooo 36	timestamp = 28160, pkt len = 61
Seq num 38 ooo 37	timestamp = 28480, pkt len = 61
Seq num 40 ooo 38	timestamp = 29120, pkt len = 61
Seq num 42 ooo 39	timestamp = 29760, pkt len = 61
Seq num 44 ooo 40	timestamp = 30400, pkt len = 61
Seq num 46 ooo 41	timestamp = 31040, pkt len = 61
Seq num 36 ooo 42	timestamp = 27840, pkt len = 61
Seq num 48 ooo 43	timestamp = 31680, pkt len = 61
Seq num 39 ooo 44	timestamp = 28800, pkt len = 61
Seq num 41 ooo 45	timestamp = 29440, pkt len = 61
Seq num 50 ooo 46	timestamp = 32320, pkt len = 61
Seq num 53 ooo 47	timestamp = 33280, pkt len = 61
Seq num 55 ooo 48	timestamp = 33920, pkt len = 61
Seq num 43 ooo 49	timestamp = 30080, pkt len = 61
Seq num 57 ooo 50	timestamp = 34560, pkt len = 61

⋮

Packet Log Excerpt

Ingress Packet info for SSRC = 0xbad52e64, first seq num = 3, last seq num = 651 ...

```
Seq num 4 ooo 3      timestamp = 1280, pkt len = 6 SID
Seq num 3 ooo 4      timestamp = 960,  pkt len = 61
Seq num 5            timestamp = 3840, pkt len = 6 SID
Seq num 6            timestamp = 6400, pkt len = 6 SID
Seq num 7            timestamp = 8960, pkt len = 6 SID
Seq num 8            timestamp = 11520, pkt len = 6 SID
Seq num 9            timestamp = 14080, pkt len = 6 SID
Seq num 10           timestamp = 16640, pkt len = 6 SID
Seq num 12 ooo 11    timestamp = 18560, pkt len = 61
Seq num 15 ooo 12    timestamp = 19520, pkt len = 61
Seq num 11 ooo 13    timestamp = 20240, pkt len = 61
Seq num 13 ooo 14    timestamp = 18880, pkt len = 61
Seq num 14 ooo 15    timestamp = 19200, pkt len = 61
Seq num 18 ooo 16    timestamp = 20480, pkt len = 61
Seq num 19 ooo 17    timestamp = 20800, pkt len = 61
Seq num 16 ooo 18    timestamp = 19840, pkt len = 61
Seq num 21 ooo 19    timestamp = 21440, pkt len = 6 SID
Seq num 23 ooo 20    timestamp = 23680, pkt len = 61
Seq num 24 ooo 21    timestamp = 24000, pkt len = 61
Seq num 25 ooo 22    timestamp = 24320, pkt len = 61
Seq num 27 ooo 23    timestamp = 24960, pkt len = 61
Seq num 28 ooo 24    timestamp = 25280, pkt len = 61
Seq num 31 ooo 25    timestamp = 26240, pkt len = 61
Seq num 32 ooo 26    timestamp = 26560, pkt len = 61
Seq num 34 ooo 27    timestamp = 27200, pkt len = 61
Seq num 17 ooo 28    timestamp = 20160, pkt len = 61
Seq num 20 ooo 29    timestamp = 21120, pkt len = 61
Seq num 22 ooo 30    timestamp = 23360, pkt len = 61
Seq num 26 ooo 31    timestamp = 24640, pkt len = 61
Seq num 29 ooo 32    timestamp = 25600, pkt len = 61
Seq num 30 ooo 33    timestamp = 25920, pkt len = 61
Seq num 33 ooo 34    timestamp = 26880, pkt len = 61
Seq num 35           timestamp = 27520, pkt len = 61
Seq num 37 ooo 36    timestamp = 28160, pkt len = 61
Seq num 38 ooo 37    timestamp = 28480, pkt len = 61
Seq num 40 ooo 38    timestamp = 29120, pkt len = 61
Seq num 42 ooo 39    timestamp = 29760, pkt len = 61
Seq num 44 ooo 40    timestamp = 30400, pkt len = 61
Seq num 46 ooo 41    timestamp = 31040, pkt len = 61
Seq num 36 ooo 42    timestamp = 27840, pkt len = 61
Seq num 48 ooo 43    timestamp = 31680, pkt len = 61
Seq num 39 ooo 44    timestamp = 28800, pkt len = 61
Seq num 41 ooo 45    timestamp = 29440, pkt len = 61
Seq num 50 ooo 46    timestamp = 32320, pkt len = 61
Seq num 53 ooo 47    timestamp = 33280, pkt len = 61
Seq num 55 ooo 48    timestamp = 33920, pkt len = 61
Seq num 43 ooo 49    timestamp = 30080, pkt len = 61
Seq num 57 ooo 50    timestamp = 34560, pkt len = 61
```

High amount of ooo (out-of-order) example

Functionality – Frame Processing

- **Decoded packet audio data**

- buffered as frames (see Overview diagrams)

- signal processing

- **Media quality – frame level**

- “stream groups” can be defined for streams related in some way

- per-stream correction for overrun, underrun, gaps, bursts

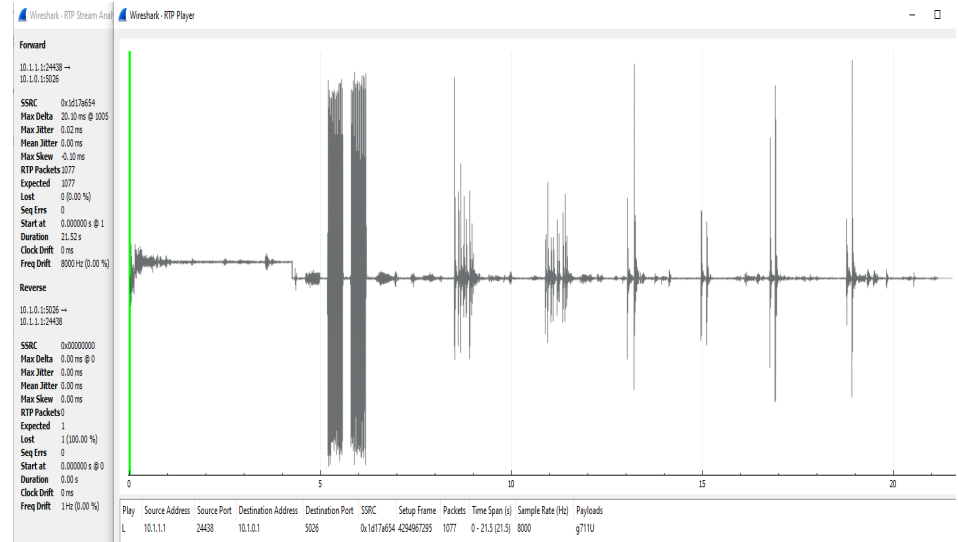
- accurate time-aligned merging / mixing of multiple endpoints

- **Real-time output streaming**

- some applications require real-time output, either per-stream or merged between related streams, typically in G711 format

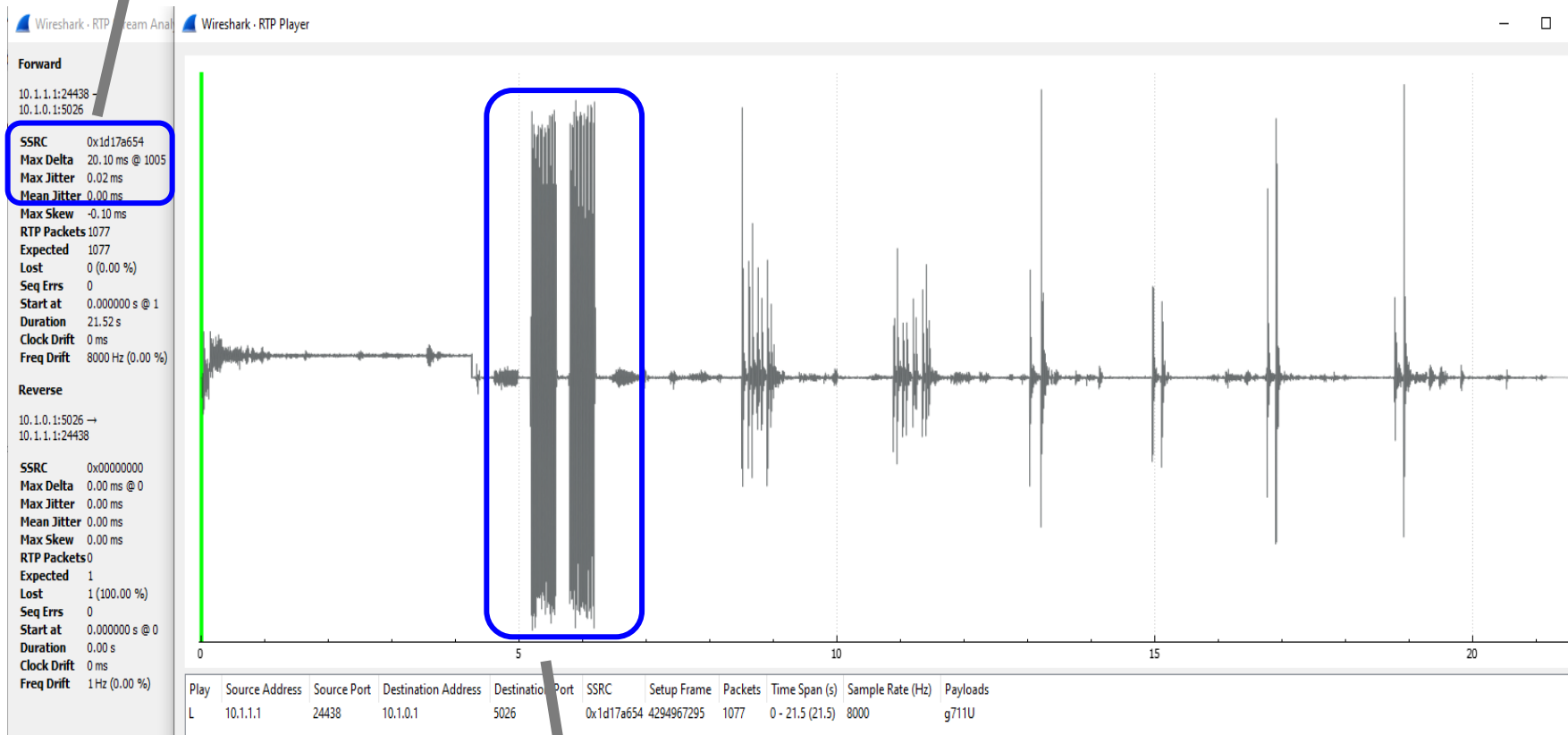
- high intelligibility required – all streams fully merged (non-overlapped) and non-duplicated *as if all endpoints are in the same room*

- **High capacity – multiple concurrent streams**



Real-Time Streaming Output Example

Reliable packet delta, no jitter over 1000s of hours of streaming



Child streams example - early media (ring tones)

Functionality – Applications

- **Dynamic and static session creation**
 - sessions created and codecs detected on-the-fly using (i) RTP only (ii) SIP invite packets (iii) .sdp files, or pre-set using static session config files
 - RTP only uses heuristic codec type detection
- **Packet push/pull interface to SigSRF libs**
 - reference application examples
 - Packet pull includes transcoded output, real-time streaming output
- **Event logging**
 - critical, major, minor, info, debug levels
 - includes alerts for thread pre-emption, queue starvation, and other performance / data related conditions
 - per-stream stats (i) on-demand, (ii) when streams close
- **Arrival timestamp reconstruction**
 - if needed due to missing / damaged arrival timestamps
 - algorithms based on queue balancing, decoded frame rate estimation

Event Log Example

```
00:00:00.000.011 INFO: DSConfigPktlib() uflags = 0x7
  P/M thread capacity max sessions = 51, max groups = 17
  Event log path = openli-voip-example_event_log_am.txt, uLogLevel = 8, uEventLogMode = 0x32, flush size = 1024, max size not set
  Debug uDebugMode = 0x0, uPktStatsLogging = 0xd, uEnableDataObjectStats = 0x1
  Screen output uPrintfLevel = 5, uPrintfControl = 0
  Energy saver p/m thread energy saver inactivity time = 30000 msec, sleep time = 1000 usec
  Alarms DSPushPackets packet cutoff alarm elapsed time not set, p/m thread preemption alarm elapsed time = 40 (msec)
00:00:00.000.721 INFO: DSConfigVoplib() voplib and codecs initialized, flags = 0x1d
00:00:00.000.749 INFO: DSConfigStreamlib() stream groups initialized
00:00:00.000.834 INFO: DSAssignPlatform() system CPU architecture supports rdtscp instruction, TSC integrity monitoring enabled
00:00:00.000.953 INFO: DSOpenPcap() opened pcap input file: ../pcaps/openli-voip-example.pcap
00:00:00.008.396 INFO: DSConfigMediaService() says setpriority() set Niceness to -15 for pkt/media thread 0
00:00:00.008.418 INFO: initializing packet/media thread 0, uFlags = 0x1180101, threadid = 0x7f320f34a700, total num pkt/med threads = 1
00:00:00.058.474 mediaMin INFO: SIP invite found, dst port = 43333, pyld len = 1994, len = 717, rem = 1979, start = 8, index = 0
o=02825591554 0 0 IN IP4 192.168.1.73
c=IN IP4 192.168.1.73
m=audio 5000 RTP/AVP 9 0 8 101
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=extmap:1 urn:ietf:params:rtp-hdrex:csrc-audio-level
a=zrtp-hash:1.10 1c812535e276bf518418c4146a20fd56e715704da9c591ae32d58ee6fed6d40f
m=video 5002 RTP/AVP 96 99
a=recvonly
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4DE01f;packetization-mode=1
a=imageattr:96 send * recv [x=[0-1920],y=[0-1080]]
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4DE01f
a=imageattr:99 send * recv [x=[0-1920],y=[0-1080]]
a=zrtp-hash:1.10 c1a98e15f12937b9cad2488c6091468f7610efeeefa59863c77d827669b913f38
00:00:00.058.644 INFO: DSFindDerStream() found HI interception point ID 10g-dev1, tag = 0x86, len = 8, dest port = 43332, pvld len = 1448, pvld ofs = 52
00:00:00.058.727 mediaMin INFO: Creating dynamic session 1, input #1, SDP specified codec type G711a, auto-detected bitrate 64000, stream group openli-voip-example. Creation packet info: IP ver 4, ssrc = 0x14a50012, seq num = 32584, payload type 8, pkt len 200, RTP payload size 160, cat 0
00:00:00.058.781 INFO: DSCreateSession() created stream group "openli-voip-example", idx = 0, owner session = 0, status = 1
```

Dynamic session creation

Functionality – Codecs

- **Multithreaded**

- original 3GPP source modifications
 - instance create, delete, modify implemented using XDAIS standard
 - global data moved into per-instance “state structs”
- API interface
 - voplib shared object (.so) library, C/C++ applications include voplib.h
 - DSCodecCreate returns a codec handle, usable with DSCodecEncode and DSCodecDecode
 - also with various codec-related APIs. Some examples:
 - DSGetCodecSampleRate, DSGetCodecBitRate, DSGetCodecRawFrameSize, DSGetCodecCodedFrameSize, DSGetCodecInfo, DSGetSampleRateValue, DSGetPayloadSize, etc

- **Optimization**

- compiler optimizations
- pragmas
- XDAIS standard requires all memory allocation done up-front, so no real-time mallocs or spin-locks

- **Testing**

- unit / functional testing – mediaTest app, with audio I/O (wav and other audio format files, USB audio)
- capacity / stress testing – mediaMin app, with application packet push/pull APIs, pcap files, UDP port I/O
- system testing – using mediaMin app, highlighted in “Overview” slides
- bit-exactness testing – comparison of floating-point reference vectors

Functionality – Codec API

```
/* codec instance definitions and APIs */

HCODEC DSCodecCreate(void* pCodecInfo, unsigned int uFlags); /* if DS_CC_USE_TERMINFO flag is given, pCodecInfo is interpreted as TERMINATION_INFO* (shared_include/session.h), otherwise as CODEC_PARAMS* (above) */

void DSCodecDelete(HCODEC hCodec);

int DSCodecEncode(HCODEC      hCodec,
                 unsigned int  uFlags,
                 uint8_t*      inData,
                 uint8_t*      outData,
                 uint32_t      in_frameSize,
                 CODEC_OUTARGS* pOutArgs);

int DSCodecDecode(HCODEC      hCodec,
                 unsigned int  uFlags,
                 uint8_t*      inData,
                 uint8_t*      outData,
                 uint32_t      in_frameSize, /* in bytes */
                 CODEC_OUTARGS* pOutArgs);

typedef struct { /* CODEC_ENC_PARAMS */

/* generic items */

int bitrate;
int samplingRate; /* most codecs are based on a fixed sampling rate so this is used only for advanced codecs such as EVS and Opus */
float frameSize; /* amount of data (in msec) processed by the codec per frame, for example 20 msec for AMR or EVS, 22.5 msec for MELPe, etc */

:
:

/* EVS, Opus, other advanced codec items */

int sid_update_interval; /* interval between SID frames when DTX is enabled */
int rf_enable; /* channel-aware mode (for EVS only supported at 13.2 kbps) */
int fec_indicator; /* for EVS, LO = 0, HI = 1 */
int fec_offset; /* for EVS, 2, 3, 5, or 7 in number of frames */
int bandwidth_limit; /* for EVS, typically set to SWB or FB */

:
:

} CODEC_ENC_PARAMS;

typedef struct { /* CODEC_DEC_PARAMS */

/* generic items */

int bitrate; /* bitrate may not be used for codecs that can derive it from payload contents */
int samplingRate; /* not used for most codecs */
float frameSize; /* amount of data (in msec) processed by the codec per frame, for example 20 msec for AMR or EVS, 22.5 msec for MELPe, etc */

:
:

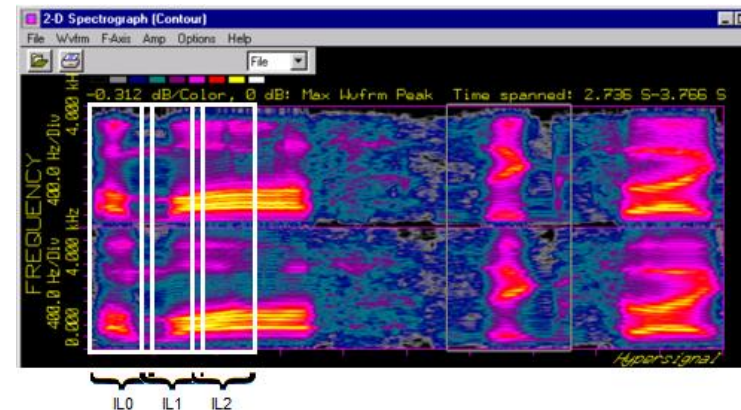
} CODEC_DEC_PARAMS;
```

• voplib.h

- excerpt shown here
- available on Github page
- C/C++ compatible

Functionality - Customer-Specific

- **Customers often ask us to incorporate / develop specific signal processing. Some examples:**
 - “deduplication” due to multiple copies of the same endpoint (with different latencies)
 - removing room echo / reverb
 - reducing background noise
- **Typically a substantial impact on performance**
- **Speech recognition (ASR)**
 - training is ultra sensitive to small changes in audio characteristics
 - production systems are trained with wide variety of “augmentations”, including background noise and babble, loud and quiet speech, frequency warping, etc.
 - preprocessing to normalize speech input decreases reliance on augmentation training and increases accuracy
 - major impact on performance; for real-time applications, concurrent streams may be reduced 10x



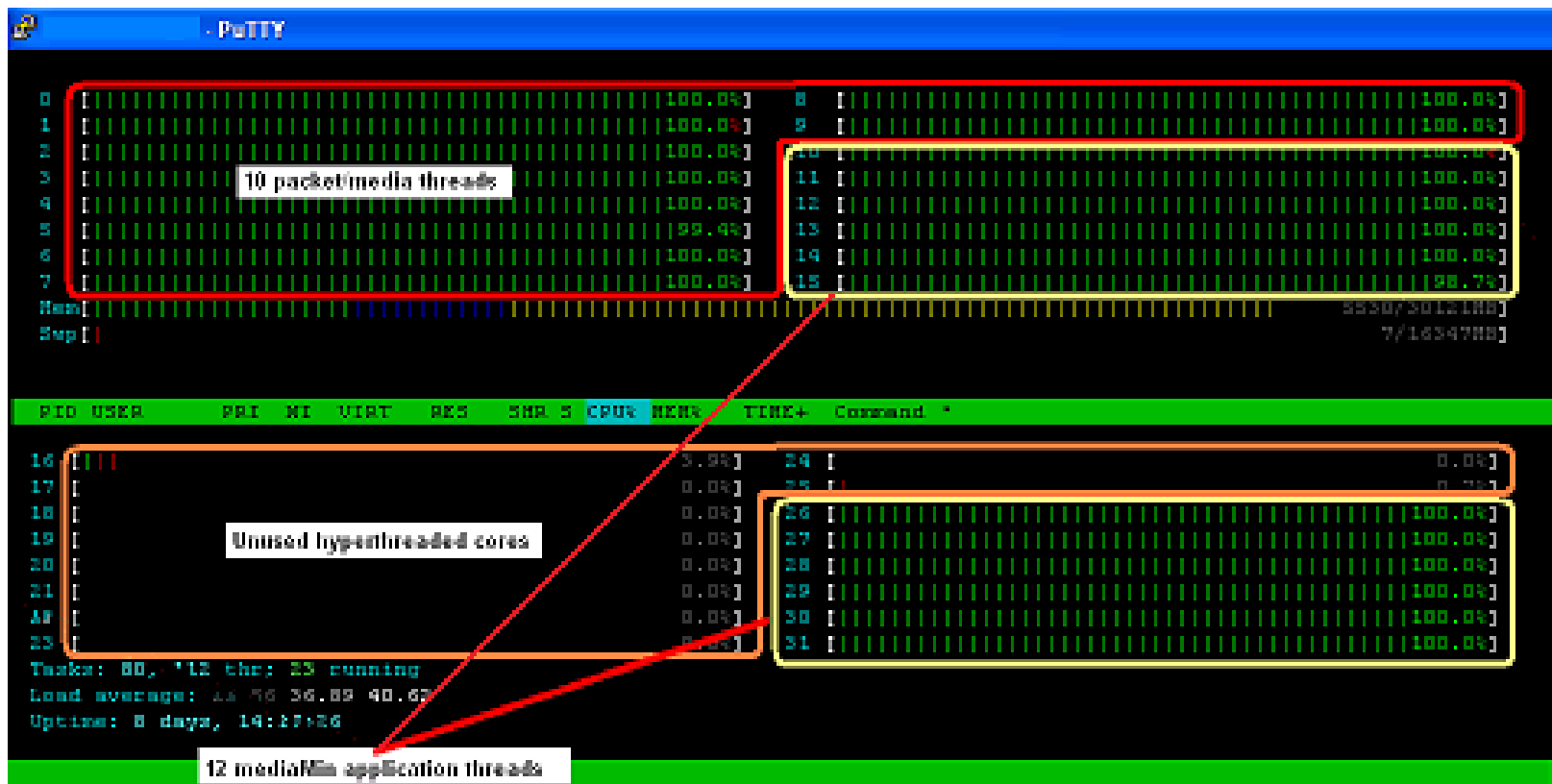
Capacity

- **Performance optimized per box / VM / container**
 - for specified core type and clock rate, we spec a max number of concurrent streams per core. For codecs sample rate and bitrate also specified
 - extensive use of htop to analyze and verify
 - we observe telecom norms – Signallogic has a long history of applications coded for high capacity, real-time performance
- **Codecs**
 - in addition to core type and clock rate, sample rate and bitrate must also be specified
 - https://www.signallogic.com/evs_codec has a Capacity Figure table for EVS on x86

Capacity, cont.

- **Extensive use of htop and to analyze / debug core usage**
 - hyperthreading must be disabled
 - stream groups must not cross core boundaries
 - look for memory leaks

htop screen capture showing
■ packet/media threads
■ application threads
■ disabled hyperthread cores



Capacity, cont.

- **Optimized for Linux**

- Linux poses performance challenges - not deterministic, not an RTOS
- carriers and LEAs understand “software defined solutions” are not deterministic, but still expect high capacity / reliability
- software detects and alarms “thread preemption” – possible performance impairment due to Linux housekeeping and other user applications

- **GPUs and DPDK ¹ may or may not be helpful**

- GPUs

- don't help with packet processing
- only “matrix expressible” operations can be easily accelerated
- can help with some codecs, but accelerating an entire codec is labor-intensive and requires hand-coding

- DPDK

- useful when combined with high-rate packet I/O hardware
- for PCIe accelerator cards, each x86 core needs a dedicated lane to avoid thread locks

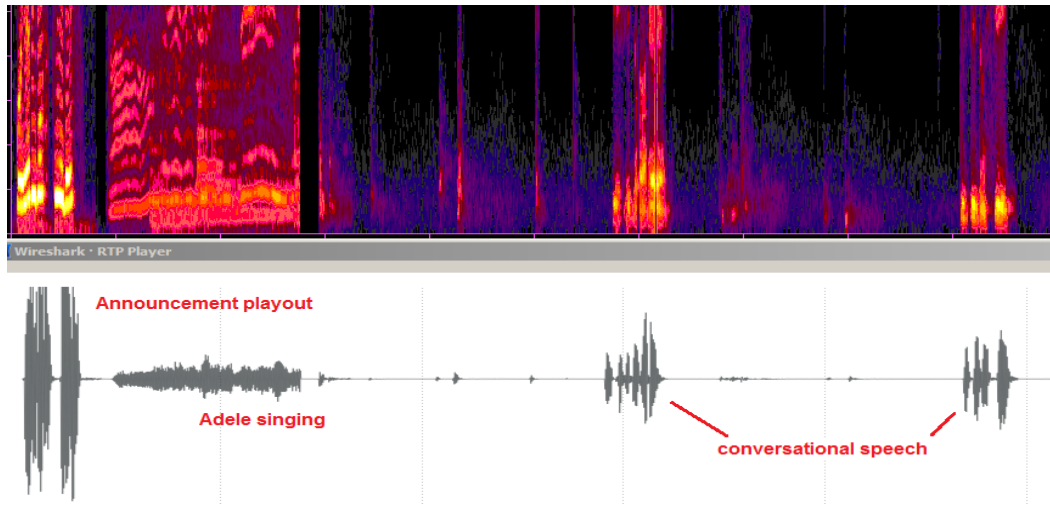
¹ Data Plane Development Kit – refers to non-Linux x86 cores dedicated to packet processing

Reliability and Testing

- **Carriers and LEAs obsess about reliability**
 - very long calls are common. All possible packet and audio data buffers and wrap conditions that could occur must be tested
 - as with capacity, we pay attention to telecom requirements. “5 9s” up time is a minimum
- **Customers run stress tests for weeks at a time**
 - we run stress tests continuously for 6+ months
 - tests include pcaps with artificial wraps, 10x packet push rates, deliberate thread preemptions, more
 - tests run at max capacity ratings
 - currently we run tests on Ubuntu 12.04 gcc++ 4.6.4 thru 20.04 gcc++ 9.3.0. Testing can be provided on CentOS systems as needed
- **Extensive use of htop and valgrind**
 - thorough and painstaking search for memory leaks
- **Software is designed for high reliability**
 - profiling and performance monitoring
 - alarms include data flow anomalies, thread preemption
 - event and packet logging
 - telemetry

Audio Quality

- **Certain customers obsess over audio quality**
 - we have observed customers using metronomes and whale sounds to verify timing and frequency integrity when testing endpoints
- **“No sound left behind”**
 - we enhance audio quality by detecting and repairing:
 - packet problems (lost packets, out-of-order, gaps, bursts)
 - stream timing (overrun, underrun, child streams)
- **Debug capability to identify root cause (CSP, cloud, or vendor)**
 - audio quality is complex and subjective; ability to identify root cause is crucial



Frequency domain analysis
and corresponding Wireshark
packet capture

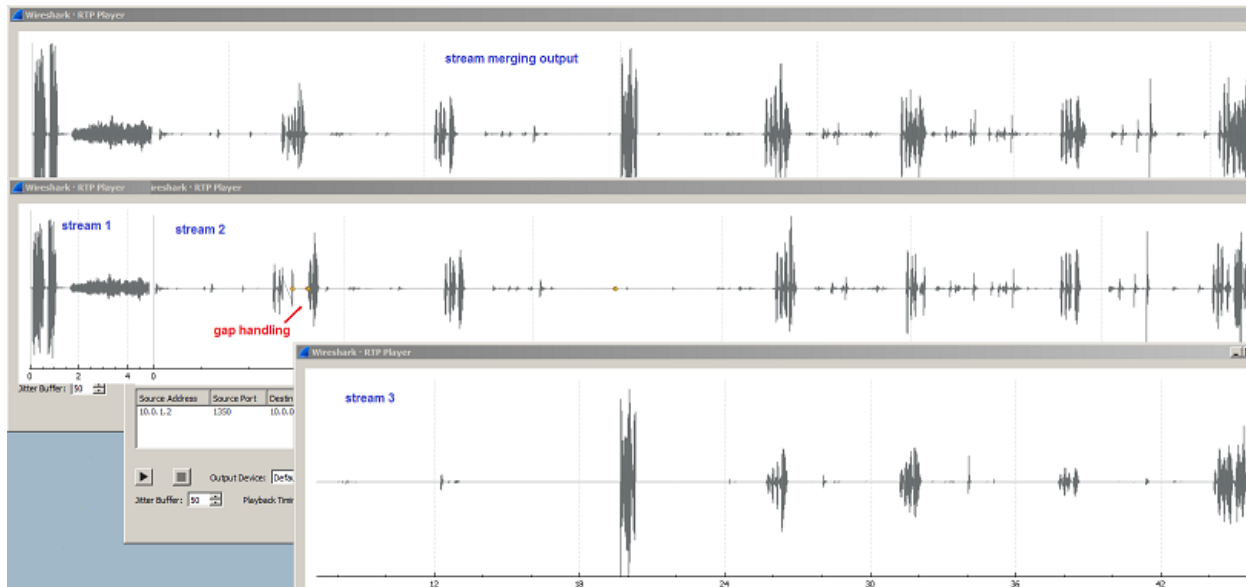
Audio Quality Challenges

- **Encapsulation artifacts**

- encapsulation packet rate may be very different than original audio RTP packet rate - slow, fast, variable. We've seen up to $\pm 15\%$
- extreme bursts of ooo (out-of-order) packets, 20-50 packets not uncommon

- **Streams not time-aligned**

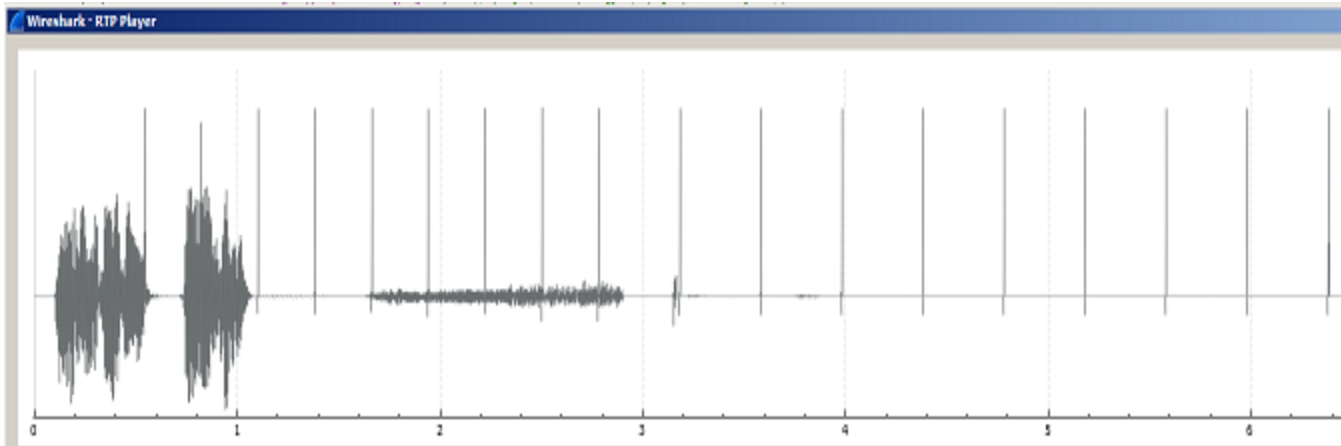
- artifacts and child streams distributed unevenly between streams
- media playout servers are particularly bad offenders



Multiple Wireshark captures showing stream merging of 3 endpoints

- **Test case verification**

- analysis and debug tools can pinpoint whether it's CSP, cloud, or handset issue
- visual audio markers can be enabled to verify timing, frame repair, etc. Different types of markers are supported



Wireshark screen capture showing audio markers inserted by software

- **Content analysis and signal processing**
 - artifact detection
 - background noise reduction
 - detecting and avoiding conversation overlap (correcting time alignment between streams in a stream group)
 - stream deduplication
- **Content recognition**
 - speech recognition
 - speaker identification
 - we use Kaldi open source
 - requires tradeoff between capacity and real-time processing
- **RTP malware detection**
 - malware payloads can hide in codec packets
 - no way to differentiate “ordinary bad voice” from “deliberate bad voice” without extensive analysis of fully decoded packets

- **Edge Computing**

- ongoing PoCs and LF Edge blueprints demonstrating hybrid cloud, enhanced privacy / security
- many telecom carriers do not trust security in public clouds

- **ASR (Automatic Speech Recognition)**

- can be done in real-time, but substantially less capacity
- not yet in real-time: individual speaker identification and transcription, known as “diarization”
- potential to reduce workloads, accurately alert on “conversations of interest”
- open source accuracy only a few % WER² more than proprietary code bases

- **Telecom migration to public cloud**

- containerized solutions needed
- LI is a particular problem due to encryption requirements
- allow CICD¹, for example improving ASR accuracy with “on the fly” training based on collected data

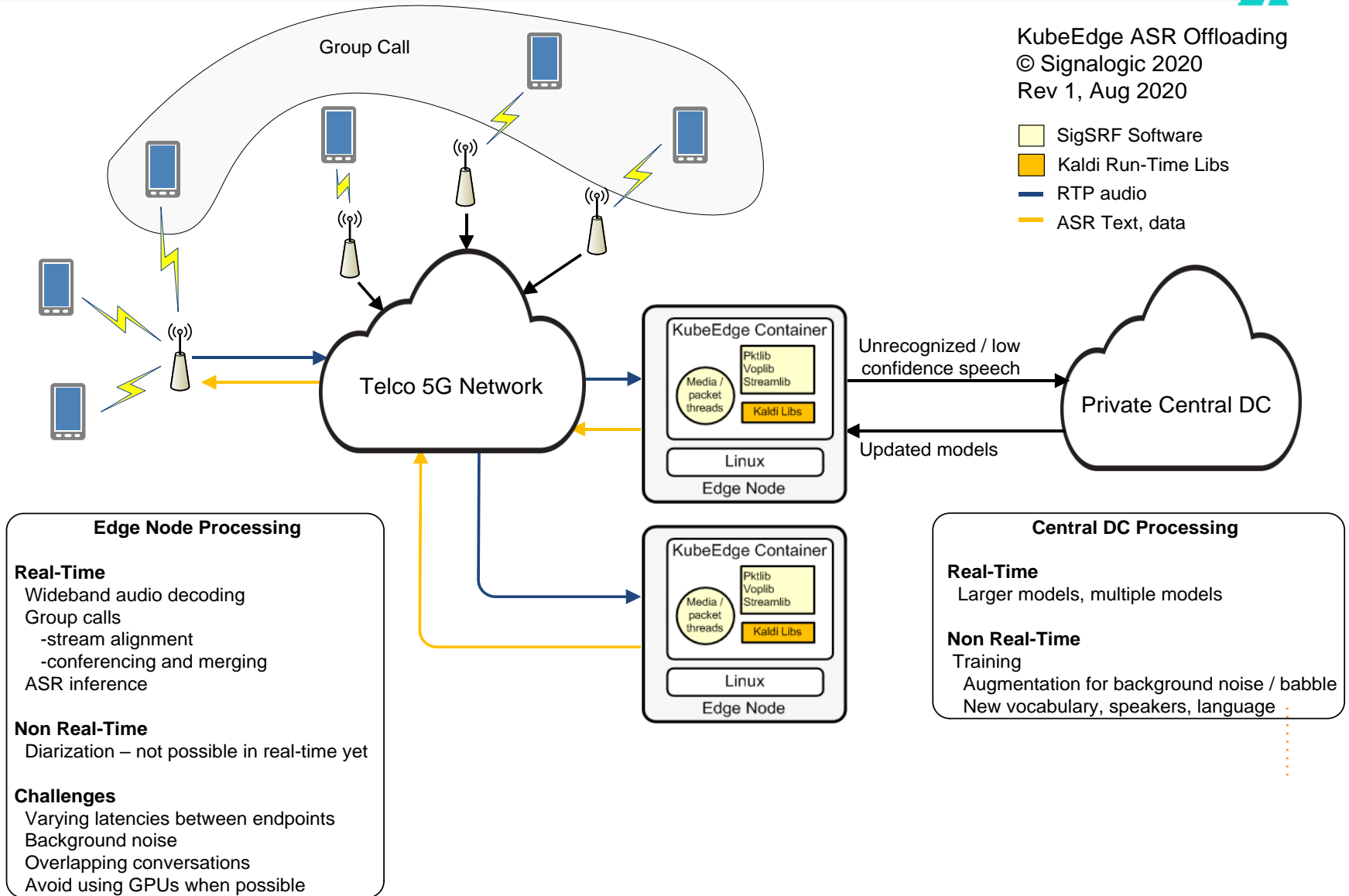
¹ Continuous Integration, Continuous Deployment

² WER = Word Error Rate

Edge Computing + Containerization

KubeEdge ASR Offloading
 © Signallogic 2020
 Rev 1, Aug 2020

- SigSRF Software
- Kaldi Run-Time Libs
- RTP audio
- ASR Text, data



Edge Node Processing

Real-Time

- Wideband audio decoding
- Group calls
 - stream alignment
 - conferencing and merging
- ASR inference

Non Real-Time

- Diarization – not possible in real-time yet

Challenges

- Varying latencies between endpoints
- Background noise
- Overlapping conversations
- Avoid using GPUs when possible

Central DC Processing

Real-Time

- Larger models, multiple models

Non Real-Time

- Training
- Augmentation for background noise / babble
- New vocabulary, speakers, language

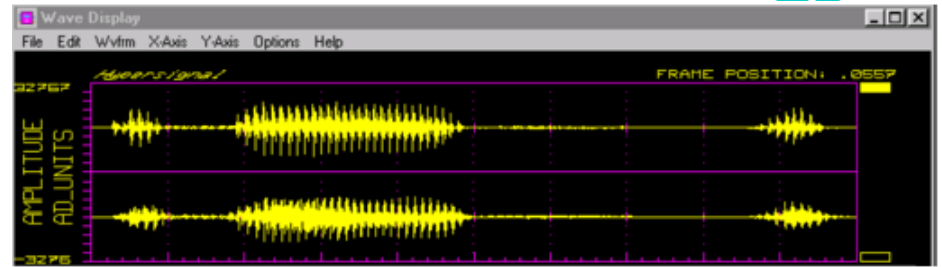
ASR Basics

- **Deep Learning Architecture**

- combines previous generation xMM¹ technology with DNNs (Deep Neural Networks)
- relies on extensive training and “augmentation” methods
- Kaldi open source is basis for Alexa, Google Home, and Cortana

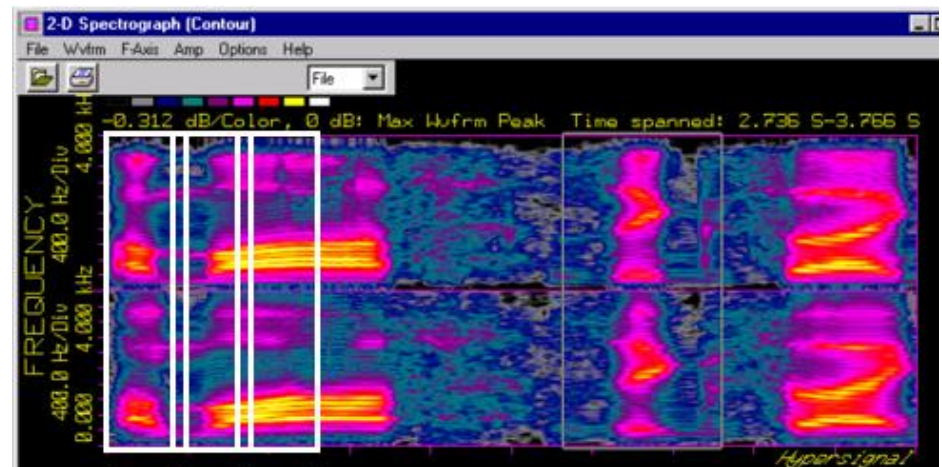
- **Frequency domain “images”**

- formed by sliding FFT analysis of incoming time series data. Each FFT frame output is similar to cochlea in human ears
- groups of FFT frames form images
- successive images are called “TDNN” (time delayed DNN), similar to series of CNNs²



Sliding FFT

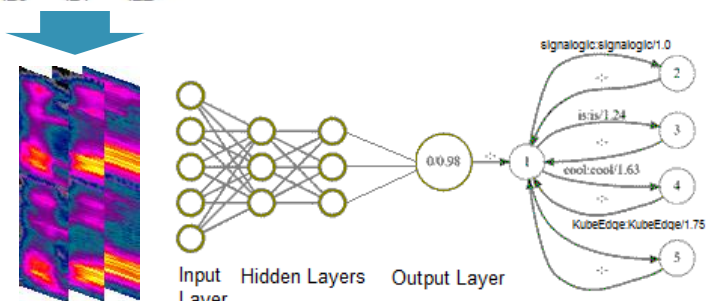
time domain (time series)



frequency domain

IL0 IL1 IL2

DNN Input Layers (ILn)



¹ Hidden Markov Model, Gaussian Mixed Model, ² Convolutional Neural Network

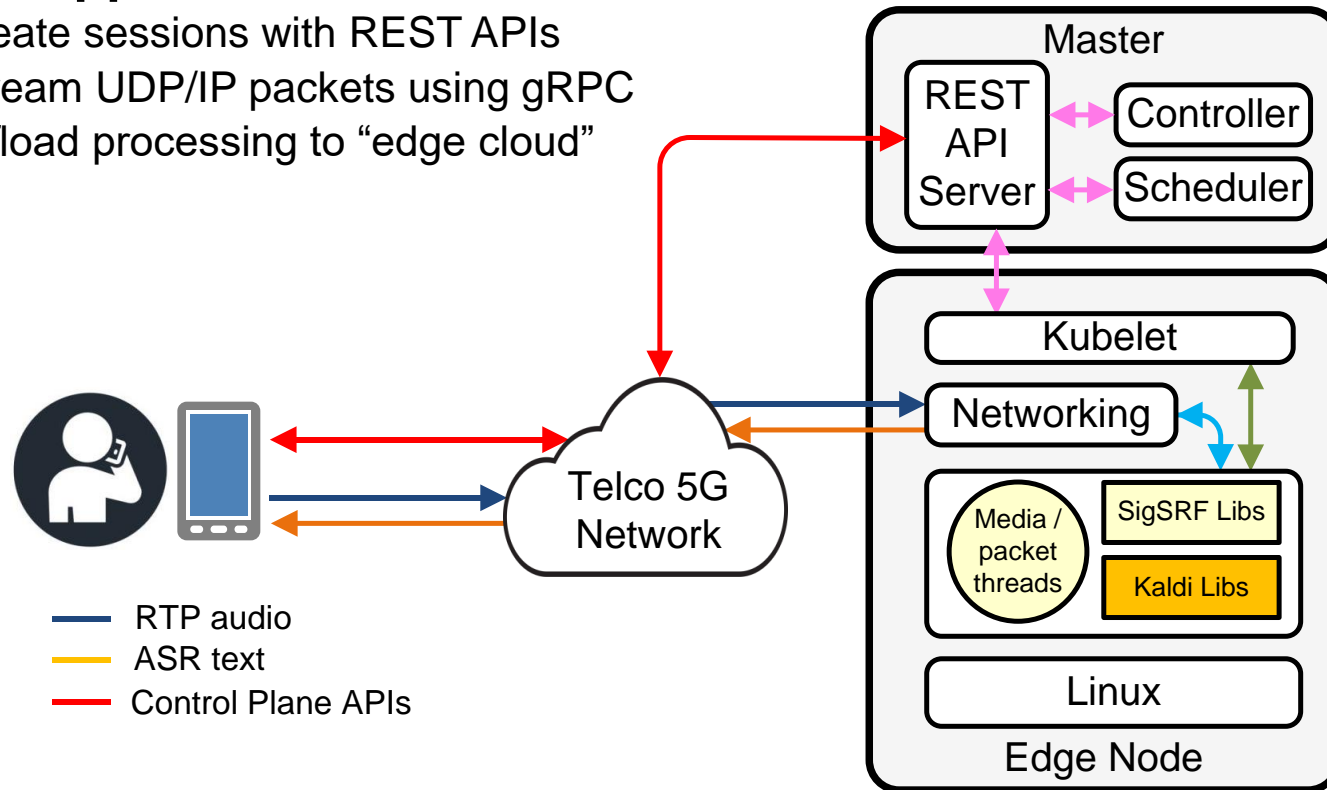
Containers and Kubernetes

- **Packet + media + ASR inside container**

- minimum 2 x86 cores, 32 GB mem, 1 TB HDD can handle 32 sessions
- a session is wideband decode (e.g. EVS), jitter buffer, stream merging up to 8 stream groups, G711 pcap output, wideband wav file output
- scales up linearly with more cores

- **Field apps**

- create sessions with REST APIs
- stream UDP/IP packets using gRPC
- offload processing to “edge cloud”



LI Perspective

- **ETSI LI Terminology: CC mediation (communication content), HI2 and HI3 (Handover Interfaces)**
- **Packet Handling**
 - Jitter buffer, packet repair, rate adjustment
- **Media**
 - Decoding (AMR, AMR-WB, EVS, more), stream alignment
- **Signal Processing**
 - Stream merging, conferencing, speech recognition

ETSI Diagram – Vendor Software Mapping
 © Signallogic 2019-2020
 Rev 1 Jun 2019

