Contents

• **Edge Computing Taking Shape**
  – Decentralization of the cloud
  – Existing problems

• **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

• **Use Cases**
  – Remotely operated and automated factory and construction equipment
  – Robotics
  – Vehicle automation
  – Malware detection

• **EdgeStream™**
  – Optimized on per thread basis, one thread per core, no spinlocks
  – Packet processing, media codecs, signal processing, inference
  – Comparison with DeepStream and GStreamer

• **Deployments**

• **Github and Docker Hub**
  – Demos, reference apps, ready-to-run containers, command lines
  – Source code
Edge Computing Taking Shape

- Densification of networks
- Decentralization of the cloud

<table>
<thead>
<tr>
<th>Legacy</th>
<th>Existing</th>
<th>Forming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hub and Spoke</td>
<td>Distributed</td>
<td>Cloud and Edge</td>
</tr>
</tbody>
</table>

User to Application Distance / Latency

- Legacy: ~10^3 km, ~10^{-1} sec
- Existing: ~10^2 km, ~10^{-2} sec
- Forming: ~10^1 km, ~10^{-3} sec

Data Flow

Pictures above copyright Equinix and Oleg Berzin, 2021-2022
Existing Problems

• Cloud computing is not designed for the edge
  – higher performance = more servers, more energy, more space
  – “economy of scale” programming model = bloatware

• Latency continues to decrease, but no real-time guarantee

• A “long supply chain” of zones, regions, services, etc. from edge to cloud
  – less secure – target rich for malware
  – edge data is private data! Images and audio may contain identifying / proprietary information. **Not to be sent to the cloud**
Existing Problems, cont.

• 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, privacy, and simplicity
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”
Use Cases

• Robotics
  – enable “no app no touch” user interfaces with ASR (automatic speech recognition) and vision

• Factory, construction, and retail – remotely controlled and automated equipment
  – accident mitigation - avoid tip-over, collision, instability
  – never execute cloud commands that contradict measurable physical data

• Vehicle automation
  – add-on small server consumer products - DIY Lidar, plate and signage recognition, smart dashcams

• Malware detection
  – detect malware payloads hiding in RTP packets

Ongoing customer contracts / discussions
EdgeStream

- **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
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**
  - also includes encoders

- **Signal processing**
  - more user-defined insertion points

DeepStream is a trademark of Nvidia Corp
Comparison with GStreamer

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

- **Packet Processing**
  - EdgeStream provides telecom grade packet processing, including …
    - loss repair
    - 500+ out-of-order handling
    - supports encapsulated protocols
    - multiple RFCs
    - logging
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/](https://openli.nz/)
Github and Docker Hub

**Github**
- SigSRF software page: [https://github.com/signalogic/SigSRF_SDK](https://github.com/signalogic/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/signalogic](https://hub.docker.com/u/signalogic)

**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@signalogic.com
- Web page: https://signalogic.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
Overview – Pipeline Flow

- Decoders
  - H.264, EVS, AMR, etc

- Input Packet Buffers

- Packet Processing
  - Repair (loss, time stamps, out-of-order), Filtering, RFC8108, DTX handling

- Decoders
  - H.264, EVS, AMR, etc

- Pre Signal Processing
  - Fs Conv, media quality enhncmt

- Per Channel Buffers

- Post Signal Processing
  - Stream merge, frame rate adjustment, Fs Conv, FFT, etc

- User Defined Processing

- AI Processing
  - ASR, CNN

- Output Packet Buffers

- Codecs
  - H.264, EVS, AMR, etc

- Network / cloud

- APIs

- Camera

- File

- Wave files

- Text

- SigSRF Multithreaded Pipeline

- I/O Media Streams
Overview – Per Thread Data Flow

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

Codecs

- pkttlib
- voplib
- streamlib

† Gap management, FLC (frame loss concealment)
‡ Includes user-defined signal processing
§ Automatic Speech Recognition

Applications
- user-defined
- mediaMin and mediaTest reference apps

Packet Loss Monitor

Input packets may be:
(i) buffered directly by p/m threads
(ii) pushed to input queues by applications

Packet / Media Threads

Output packets may be:
(i) sent directly by p/m threads
(ii) pulled from output queues by applications

 FFT

Jitter Buffer

DTS

Stream Groups

Output Packet Queues

Transcoded 0..N-1
Stream Groups 0..M-1

Codecs

Packet Loss Monitor

Input Packet Queues

Stream 0
Stream 1

Gap Management, FLC (frame loss concealment)

Includes user-defined signal processing

Automatic Speech Recognition

RFC8108 channel creation

RTMP

Jitter Buffer Add

Jitter Buffer Get

DTX

Decode

Fs

Conv

Media Domain Processing

AsR

Session Data Structs

SESSION_DATA
TERMINATION_INFO
media attributes

Transcoded 0..N-1

Stream Groups 0..M-1

Jitter Buffer 0..N-1

Encode

Output Packet Queues

UDP

Input packets may be:
(i) buffered directly by p/m threads
(ii) pushed to input queues by applications

Packet Loss Monitor

DDSTPackets()

Pcap Files
TCP
UDP

Session Data Structs
SESSION_DATA
TERMINATION_INFO
media attributes

UDP

Text or analytics output

wav file, other audio format files

Compressed data files

Output packets may be:
(i) sent directly by p/m threads
(ii) pulled from output queues by applications

Applications
- user-defined
- mediaMin and mediaTest reference apps
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
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 = 1920, pkt len = 61
Seq num 5              timestamp = 3840, pkt len = 6 SID
Seq num 6              timestamp = 4600, pkt len = 6 SID
Seq num 7              timestamp = 6384, pkt len = 6 SID
Seq num 8              timestamp = 7160, pkt len = 6 SID
Seq num 9              timestamp = 8940, pkt len = 6 SID
Seq num 10             timestamp = 10720, pkt len = 6 SID
Seq num 12 ooo 11      timestamp = 12640, pkt len = 6 SID
Seq num 15 ooo 12      timestamp = 14480, pkt len = 6 SID
Seq num 11 ooo 13      timestamp = 16320, pkt len = 6 SID
Seq num 16 ooo 14      timestamp = 18160, pkt len = 6 SID
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 = 21680, pkt len = 61
Seq num 16 ooo 18      timestamp = 24320, pkt len = 61
Seq num 21 ooo 19      timestamp = 25440, pkt len = 61
Seq num 23 ooo 20      timestamp = 26560, pkt len = 61
Seq num 24 ooo 21      timestamp = 27680, pkt len = 61
Seq num 25 ooo 22      timestamp = 28800, pkt len = 61
Seq num 27 ooo 23      timestamp = 30240, pkt len = 61
Seq num 28 ooo 24      timestamp = 31360, pkt len = 61
Seq num 31 ooo 25      timestamp = 33120, pkt len = 61
Seq num 26 ooo 27      timestamp = 34080, pkt len = 61
Seq num 29 ooo 28      timestamp = 36800, pkt len = 61
Seq num 34 ooo 29      timestamp = 39200, pkt len = 61
Seq num 17 ooo 30      timestamp = 40320, pkt len = 61
Seq num 19 ooo 31      timestamp = 41760, pkt len = 61
Seq num 30 ooo 32      timestamp = 43680, pkt len = 61
Seq num 32 ooo 33      timestamp = 45120, pkt len = 61
Seq num 34 ooo 34      timestamp = 46080, pkt len = 61
Seq num 18 ooo 35      timestamp = 48320, pkt len = 61
Seq num 35 ooo 36      timestamp = 49440, pkt len = 61
Seq num 37 ooo 37      timestamp = 50400, pkt len = 61
Seq num 38 ooo 38      timestamp = 51200, pkt len = 61
Seq num 40 ooo 39      timestamp = 52800, pkt len = 61
Seq num 42 ooo 40      timestamp = 54720, pkt len = 61
Seq num 44 ooo 41      timestamp = 56160, pkt len = 61
Seq num 46 ooo 42      timestamp = 57440, pkt len = 61
Seq num 36 ooo 43      timestamp = 59040, pkt len = 61
Seq num 28 ooo 44      timestamp = 60480, pkt len = 61
Seq num 33 ooo 45      timestamp = 63360, pkt len = 61
Seq num 41 ooo 46      timestamp = 65520, pkt len = 61
Seq num 50 ooo 47      timestamp = 68880, pkt len = 61
Seq num 53 ooo 48      timestamp = 73776, pkt len = 61
Seq num 43 ooo 49      timestamp = 75600, pkt len = 61
Seq num 57 ooo 50      timestamp = 80160, 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: initial packet/media thread 0, uFlags = 0x1180101, threadid = 0x7f320f34a700, total num pkt/med threads = 1
00:00:00.058.418 INFO: SIP invite found, dst port = 43333, pyld len = 1994, len = 717, rem = 1979, start = 8, index = 0
0=2825591554 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-hdrext:csrcudio-level
a=zrtp-hash:1.10 1c812535e276bf518418c4146a20fd56e715704da9c591ae32d58e6fed6d40f
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:99 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 c1a98e15f2d299b37b9cad2488c6091468f7610eefef59863c77d827666b913f38
00:00:00.058.644 INFO: DSPIndNerStream() found HI interception point ID 10g-dev1, tag = 0x86, len = 8, dest port = 43332, pyld len = 1448, pyld 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, ssr = 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
**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 – Signalogic 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.signalogic.com/evs_codec](https://www.signalogic.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
• **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

[Image showing 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 ±15%
  – extreme bursts of 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
Audio Quality Verification and Debug

• **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](image-url)
Media Content Processing

- **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
Current R&D

• **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 % WER\(^2\) more than proprietary code bases

• **Telecom migration to public cloud**
  – containerized solutions needed
  – LI is a particular problem due to encryption requirements
  – allow CICD\(^1\), 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
© Signalogic 2020
Rev 1, Aug 2020

Unrecognized / low confidence speech

Group Call

Telco 5G Network

KubeEdge ASR Offloading

Central DC Processing

Real-Time
Larger models, multiple models

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

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

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

Telco 5G Network

Linux
Edge Node

KubeEdge Container

Media/packet streams
Kaldi Libs

Private Central DC

Unrecognized / low confidence speech
Updated models

KubeEdge Container

Media/packet streams
Kaldi Libs

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

Telco 5G Network

Group Call

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
ASR Basics

- **Deep Learning Architecture**
  - Combines previous generation xMM\(^1\) 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 CNNs\(^2\)

---

\(^1\) 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
© Signalogic 2019-2020
Rev 1 Jun 2019

9Mar22
Prepared by Signalogic for Akraiino Technical Meeting 1Q22