Adding ASR to an Akraino Robot

Signalologic, Inc.
Dallas, Texas
Contents

• Executive summary
• Problems facing robotics ASR
  – 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

- **Signalogic is adding ASR (automatic speech recognition) to an Akraino robotics blueprint**
  - key blueprint organizations: Fujitsu and Ritsumeikan University
  - [https://wiki.akraino.org/display/AK/Robot+basic+architecture+based+on+SSES](https://wiki.akraino.org/display/AK/Robot+basic+architecture+based+on+SSES)

- **Real-time with a 20K word vocabulary**
  - Signalogic software enables both high performance and energy efficient implementation
  - typical robot compute resources are insufficient for both background noise removal and real-time, high accuracy, high vocabulary ASR
  - the blueprint will use a Kaldi 20K word English vocabulary

- **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.

- **Planning to include ASR in release 7 of the Fujitsu / Ritsumeikan blueprint**
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 must be continuous / fail-safe, regardless of intermittent internet connectivity

• Human safety is paramount
  – accuracy must be high – when someone says “stop” or “watch out” 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
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, especially urgent 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 with other private nodes

• **Energy and size constraints**
  – operate with 50 – 75 W power consumption with no fans
  – operate in extremely small-form factor
  – must be physically lightweight, especially for mobile robots

“Beat it roomba, come back later”
Software – EdgeStream™ Platform

• **Input from USB audio or RTP packets**
  – input from one more microphones
  – input from RTP packet streams (IP/UDP, microphones with Ethernet interface). A wide range of codecs supported (AMR, EVS, G711, etc)

• **Pre-processing (e.g. background noise removal)**
  – environment background noise
  – robot noise, e.g. motors, wheels, brushes

• **Application specific processing if needed**
  – lawful intelligence / interception
  – telecom
  – application-specific signal processing

• **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
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

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
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 fans
  – boots Ubuntu 20.04

• **Essentially a “robot server”**
  – straightforward to develop code on lab and cloud servers, then run on robot servers
  – lab / cloud servers can 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/](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

**EdgeStream™ Workflow**
© Signalogic 2021
Rev 1a, Dec 2021

- **Decoders**
  - H.264, EVS, AMR, etc.
- **Input Packet Buffers**
- **Packet Processing**
  - Repair (loss, time stamps, out-of-order), Filtering, RFC8108, DTX handling
- **Codecs**
- **Pre Signal Processing**
  - Fs Conv, media quality enhancement
- **Per Channel Buffers**
  - Frame Buffers
- **Post Signal Processing**
  - Stream merge, frame rate adjustment, Fs Conv, FFT, etc
- **User Defined Processing**
- **AI Processing**
  - ASR, CNN
- **I/O Media Streams**
  - Camera
  - File
  - Network / cloud
  - APIs
- **Per Stream Buffers**
- **Output Packet Buffers**
- **Encoders**
  - H.264, EVS, AMR, etc.
Overview – Per Thread Data Flow

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

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

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

Codecs

Packet / Media Threads

Jitter Buffer 0..N
Stream Groups 0..M
Session Data Structs
- SESSION_DATA
- TERMINATION_INFO
- media attributes

DSPullPackets() DSPCreateSession() DSPSet/GetSessionInfo() DSPDeleteSession() Session Data Structs

Applications
-user-defined
-mediaMin and mediaTest reference apps

Pktlib APIs

Packet Loss Monitor

Packet / Media Threads

Input Packet Queues

Stream 0
Stream 1
Stream N-1

Jitter Buffer Add
Jitter Buffer Get
DTX

Decode
Fs
Conv

Stream Groups - merging, conf
-audio quality

Media Domain Processing - signal processing
- ASR

Encode
Fs
Conv

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

Text or analytics output

Compressed data files

wav file, other audio format files

RFC8108

channel creation

SID repair

Jitter Buffer 0..N-1

Pcap Files
TCP
UDP

Pktlib APIs

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

<table>
<thead>
<tr>
<th>Seq num</th>
<th>Packet ID</th>
<th>Timestamp</th>
<th>Packet Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>1280</td>
<td>6 SID</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>960</td>
<td>61</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3840</td>
<td>6 SID</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4400</td>
<td>6 SID</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7200</td>
<td>6 SID</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>11520</td>
<td>6 SID</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>14080</td>
<td>6 SID</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>16640</td>
<td>6 SID</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>18560</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>19520</td>
<td>61</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>19880</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>20480</td>
<td>61</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>20800</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>21120</td>
<td>61</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>21920</td>
<td>61</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>22160</td>
<td>61</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>22320</td>
<td>61</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>22680</td>
<td>61</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>22520</td>
<td>61</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>22960</td>
<td>61</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>23380</td>
<td>61</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>23680</td>
<td>61</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>24000</td>
<td>61</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>24320</td>
<td>61</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>24680</td>
<td>61</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>24960</td>
<td>61</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>25280</td>
<td>61</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>25650</td>
<td>61</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>26240</td>
<td>61</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>26560</td>
<td>61</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>27200</td>
<td>61</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>27700</td>
<td>61</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>28160</td>
<td>61</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>28480</td>
<td>61</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>28840</td>
<td>61</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>29120</td>
<td>61</td>
</tr>
<tr>
<td>38</td>
<td>39</td>
<td>29440</td>
<td>61</td>
</tr>
<tr>
<td>39</td>
<td>40</td>
<td>29760</td>
<td>61</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>30400</td>
<td>61</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>31040</td>
<td>61</td>
</tr>
<tr>
<td>42</td>
<td>43</td>
<td>31680</td>
<td>61</td>
</tr>
<tr>
<td>43</td>
<td>44</td>
<td>32360</td>
<td>61</td>
</tr>
<tr>
<td>44</td>
<td>45</td>
<td>32960</td>
<td>61</td>
</tr>
<tr>
<td>45</td>
<td>46</td>
<td>33280</td>
<td>61</td>
</tr>
<tr>
<td>46</td>
<td>47</td>
<td>33920</td>
<td>61</td>
</tr>
<tr>
<td>47</td>
<td>48</td>
<td>34560</td>
<td>61</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>35200</td>
<td>61</td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>35920</td>
<td>61</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>36560</td>
<td>61</td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>37200</td>
<td>61</td>
</tr>
<tr>
<td>52</td>
<td>53</td>
<td>37840</td>
<td>61</td>
</tr>
<tr>
<td>53</td>
<td>54</td>
<td>38480</td>
<td>61</td>
</tr>
<tr>
<td>54</td>
<td>55</td>
<td>39120</td>
<td>61</td>
</tr>
<tr>
<td>55</td>
<td>56</td>
<td>39760</td>
<td>61</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>40400</td>
<td>61</td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>41040</td>
<td>61</td>
</tr>
<tr>
<td>58</td>
<td>59</td>
<td>41680</td>
<td>61</td>
</tr>
<tr>
<td>59</td>
<td>60</td>
<td>42360</td>
<td>61</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
<td>43040</td>
<td>61</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>43700</td>
<td>61</td>
</tr>
<tr>
<td>62</td>
<td>63</td>
<td>44360</td>
<td>61</td>
</tr>
<tr>
<td>63</td>
<td>64</td>
<td>45040</td>
<td>61</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>45700</td>
<td>61</td>
</tr>
<tr>
<td>65</td>
<td>66</td>
<td>46360</td>
<td>61</td>
</tr>
<tr>
<td>66</td>
<td>67</td>
<td>47040</td>
<td>61</td>
</tr>
<tr>
<td>67</td>
<td>68</td>
<td>47700</td>
<td>61</td>
</tr>
<tr>
<td>68</td>
<td>69</td>
<td>48360</td>
<td>61</td>
</tr>
<tr>
<td>69</td>
<td>70</td>
<td>49040</td>
<td>61</td>
</tr>
<tr>
<td>70</td>
<td>71</td>
<td>49700</td>
<td>61</td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>50360</td>
<td>61</td>
</tr>
<tr>
<td>72</td>
<td>73</td>
<td>51040</td>
<td>61</td>
</tr>
<tr>
<td>73</td>
<td>74</td>
<td>51700</td>
<td>61</td>
</tr>
<tr>
<td>74</td>
<td>75</td>
<td>52360</td>
<td>61</td>
</tr>
<tr>
<td>75</td>
<td>76</td>
<td>53040</td>
<td>61</td>
</tr>
<tr>
<td>76</td>
<td>77</td>
<td>53700</td>
<td>61</td>
</tr>
<tr>
<td>77</td>
<td>78</td>
<td>54360</td>
<td>61</td>
</tr>
</tbody>
</table>
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 = 22080, 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 = 20400, 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 26 ooo 27       timestamp = 26560, pkt len = 61
Seq num 34 ooo 28       timestamp = 27200, pkt len = 61
Seq num 17 ooo 29       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 32 ooo 26       timestamp = 25920, pkt len = 61
Seq num 34 ooo 27       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 49 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 52 ooo 47       timestamp = 33920, pkt len = 61
Seq num 54 ooo 48       timestamp = 30080, pkt len = 61
Seq num 55 ooo 49       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  DSSPushPackets 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
c02825591554 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:8 PCMA/8000
a=rtpmap:8 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=extmap:1 urn:ietf:params:rtp-hdrext:csrc-audio-level
a=zrtp-hash:1.10 c1a98e15f12937b9cad2488c6091468f7610efefea59863c77d827669b913f38
m=video 5002 RTP/AVP 96 99
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 c1a98e15f12937b9cad2488c6091468f7610efefea59863c77d827669b913f38
00:00:00.000.085.644 INFO: DSPFindInterception() found HI interception point ID 10g-dev1, tag = 0x86, len = 8, dest port = 43332, pyld len = 1448, pyld ofs = 52
00:00:00.008.727 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
00:00:00.000.781 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
    • DSCCodecCreate returns a codec handle, usable with DSCCodecEncode and DSCCodecDecode
    • 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

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
• **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. **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

  2. **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 ±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
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**

**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

---

KubeEdge ASR Offloading
© Signalogic 2020
Rev 1, Aug 2020

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

Telco 5G Network

Private Central DC

Group Call

Unrecognized / low confidence speech
Updated models

Start of diagram

End of diagram
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