High Performance Robotics Edge Platform

Safe, private, cost-effective robot solution

Signalogic, Inc.
Dallas, Texas
Contents

• **Problems and needs**
  – real-time voice commands: hands-free operation, safety / emergency concerns
  – data privacy
  – high performance onboard processing without costly cloud compute resources

• **Robotics edge platform**
  – software - EdgeStream™ platform optimized for high performance edge computing / analytics
  – hardware – extremely small form-factor, quad core server. Low SWaP ¹
  – containerized solution

• **Akraino SSES Blueprint**
  – Fujitsu + Ritsumeikan University
  – food preparation and handling robotics

• **Architecture and data flow**
  – core / thread data flow – ASR, vision, failure prediction

• **Signal processing**
  – noise reduction, sound classification
  – failure prediction

• **ASR** ²
  – urgent commands, factory hands-free environment
  – 20K word vocabulary, high accuracy

• **Roomba lab demo**
  – robotics server and battery “dead bugged” on a Roomba
  – voice commands processed and sent to Roomba APIs via USB; e.g. “watch out !”, “stop”, “turn left”, etc

¹ Size, weight, and power consumption
² Automatic speech recognition
Cloud Tradeoffs in High Performance Robotics

• **For high performance robotics the cloud …**
  – lacks provide data privacy (e.g. background conversations, proprietary sounds)
  – cannot guarantee real-time response in human safety / emergency situations
  – is cost-prohibitive as number of robots increases

• **… but the cloud enables**
  – essential microservices – location, coordination with other robots, etc.
  – communication with IoT sensors
  – containerization management and orchestration
  – app interfaces – robot status and monitoring, telemetry, event notification

*In the Roomba example, voice commands must travel to the cloud and back – slow, not private, and not reliable*
High Performance Robotics Needs

• “Laws of robotics”
  – respond to voice commands, especially urgent safety / emergency commands
  – avoid people, animals, property
  – do not break things and do not incur legal liability
  – operate with zero trust in cloud responses – ground situation takes precedence

• Cannot be cost-prohibitive to scale
  – as an example, AWS “Lightsail” compute instances cost $80 per month
    – dedicated, always available, high performance quad x86 cores with SSE
    – 8 GB mem, 320 GB SSD
  – at first glance looks great, but cost adds up for multiple robots

• Maintain strict data privacy
  – background sounds, conversation are considered strictly private
  – data may be sent to cloud for AI training, within carefully defined policies

• Ultra low SWaP
  – under 50 W power consumption with no fans
  – extremely small form-factor
  – must be physically lightweight, especially for mobile robots
RobotHPC™ Edge Platform - Software

• **EdgeStream™ software**
  – optimized for per core high performance, low energy usage
  – deployed with telecom, lawful interception, and gov/mil customers
  – using quad-core Atom, handles up to three (3) concurrent ASR streams, three (3) far-field microphones for one stream, other sensor types, or some combination

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

• **Supports cloud connectivity**
  – essential microservices – location, coordination with other robots, etc.
  – communication with IoT sensors
  – RobotHPC containerization, management and orchestration
  – app interfaces – robot status and monitoring, telemetry, event notification

• **Development environment**
  – Ubuntu 20.04, gcc/g++ 8.x and higher
  – command line, WinSCP, gdebug … standard Linux and WinX tools
RobotHPC™ Edge Platform - Hardware

- **Ultra low SWaP server**
  - *small* pico ITX form-factor, 3.5” x 3.5”
  - *high performance* quad-core Atom (x5-E3940), 8 GB mem
  - *low power* ~50 W, no fans
  - *integrated* SSD, WiFi, HDMI, USB, etc.

- **Flexible, easy-to-use, standard**
  - code developed on x86 lab / cloud servers, runs on robotics servers w/o changes. No IDEs, JTAG emulators, or special “translation tools” / “porting tools” for AI frameworks
  - standard Linux (Ubuntu 20.04), gcc/g++, debug tools
  - 100% compatible with mainstream open source edge computing and analytics
  - 100% compatible with mainstream containerization and cloud orchestration
SSES Akraino Blueprint

• Fujitsu + Ritsumeikan University
  – food preparation and handling for manufacturing and enterprise customers
  – hands-free operation – wet environment, workers wearing gloves
  – safety and emergency needs
  – robots include x86 based small servers

• Signalogic contribution
  – speech recognition – high performance, high vocabulary ASR
  – optimized solution, consumes one (1) x86 core from available pool of cores

SSES architecture diagram showing RobotHPC™ data flow
Core / Thread Data Flow

- **Pinned x86 cores**
  - noise removal, sound classification, ASR
  - failure prediction
  - TBD, as needed
  - one core reserved for Linux

- **Each core runs end-to-end thread**
  - real-time
  - including input and output

- **EdgeStream™ software**
  - makes this type of real-time architecture possible – no thread slicing
  - no spinlocks
  - data passed between cores with lock-free buffers

- Diagram showing data flow and core assignments:
  - Core 0: Linux
    - network
    - file I/O and mgmt
    - system
  - Core 1: ASR
  - Core 2: Failure Prediction
    - Gyroscope, Compass, Accelerometer (vibration)
  - Core 3: TBD
    - vision
    - additional mics
    - other sensors
  - Core 0 connected to Core 1 via USB
  - Core 1 connected to Core 2 via USB
  - Core 1 connected to Core 3 via USB
  - Fujitsu Multi-Sensor Module (MSM) connected to Core 0 via USB
  - Acoustic (sound) connected to Core 1 via USB

- Includes deep learning inference
ASR Data Flow

- **Wideband audio input**
  - 16 kHz sampling rate, 16-bit data
  - input from one or more USB microphones, A/D converters, etc

- **Frequency domain “images”**
  - formed by overlap 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 \(^3\)), similar to series of CNN \(^4\)
  - xMM \(^5\) applied to DNN outputs

---

\(^3\) Deep neural network

\(^4\) Convolutional neural network

\(^5\) Hidden Markov model, Gaussian mixed model
Signal Processing

• **Noise removal**
  – spectral subtraction and other methods minimize environment background noise
  – target robot noise, e.g. motors, wheels, brushes
  – detect and minimize incidental background conversation

• **Sound classification**
  – voice, sound, noise

• **Failure prediction**
  – incorporate multiple sensors (vibration, acceleration, acoustic)
  – apply deep learning methods
  – recognize “sequences of sounds”, similar to language sentences
ASR

• **Capabilities**
  – urgent / safety voice commands; e.g. “stop”, “watch out”
  – operating commands; e.g. “change mode”, “discard”
  – large vocabulary, context-awareness provides error resistance, reducing false positives
  – currently running in real-time on robotics edge platform in the lab (x5-E3940 Atom)

• **Deep learning architecture**
  – combines xMM and DNN technology
  – based on Kaldi open source, which is used by Alexa, Google Home, and Cortana

• **Processing intensive**
  – real-time on one (1) x5-E3940 core, but not by a large margin (RTF \(^6\) approx 2)

• **Fast development / debug / test**
  – develop, debug, and test on any x86 (with SSE), run on the robotics platform. No conversion to “internal representation”, model translation tools, etc needed

\(^6\) Real-time factor
Roomba Demo Objectives

- **Edge computing high performance**
  - real-time
  - respond to voice commands, especially urgent commands
  - vision - avoid people, pets, property. Don’t break things

- **Cloud independence**
  - operate with zero trust in cloud commands – on-the-ground situation takes precedence
  - factory background conversation, sounds are strictly private
  - cloud data storage, training only under well-defined, controlled conditions

- **Extremely low SWaP**
  - operate with ~50 W power consumption, no fans
  - ultra small form-factor
  - physically lightweight, especially suitable for mobile robots

“Not now roomba”
Roomba Lab Demo

• **Robotics edge platform**
  – “dead bugged” onto Roomba 680
  – quad-core Atom server, no fans
  – 12V dc (battery or AC adapter)
  – USB interface to Roomba RS-232
  – line-in or USB microphone

• **Lab interface**
  – standard server tools used for development, test, and debug – WinSCP, cmd line, gdebug, etc.
  – for hands-on lab work, HDMI monitor, keyboard available

• **Roomba real-time voice control**
  – real-time ASR, 20k word vocabulary
  – translate recognized ASR text to Roomba command APIs
Info & Resources: Akraino, Github, Docker Hub

- **Akraino**
  - SSES Blueprint wiki page: [https://wiki.akraino.org/display/AK/CPS+Robot+Blueprint+family](https://wiki.akraino.org/display/AK/CPS+Robot+Blueprint+family)

- **Product**
  - [https://signalogic.com/RobotHPC](https://signalogic.com/RobotHPC)

- **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/RobotHPC
Supplemental

• Following slides are background info …
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
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
Overview – Per Thread Data Flow

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

Codecs

Packet / Media Threads

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

Applications
-user-defined
-mediaMin and mediaTest reference apps

Packet Loss Monitor

Stream Groups

Media Domain Processing

Session Data Structs
-SESSION_DATA
-TERMINATION_INFO
-media attributes

Codec Interfaces

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

Jul 2022
Robotics Edge Platform - Not Under NDA
Overview – Pipeline Flow

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

Codecs

Packet Processing
Decoders
H.264, EVS, AMR, etc

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
Per Stream Buffers

Input Packet Buffers

Pre Signal Processing
Fs Conv, media quality enhnmt

SigSRF Multithreaded Pipeline

Output Packet Buffers

Camera

File

Network / cloud

APIs

wav files

text
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
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/)
Software 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
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>ooo</th>
<th>timestamp</th>
<th>pkt len</th>
<th>SID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq num 4</td>
<td>ooo 3</td>
<td>12800</td>
<td>6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 3</td>
<td>ooo 4</td>
<td>9600</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Seq num 5</td>
<td>timestamp</td>
<td>38400</td>
<td>pkt len = 6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 6</td>
<td>timestamp</td>
<td>40000</td>
<td>pkt len = 6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 7</td>
<td>timestamp</td>
<td>8600</td>
<td>pkt len = 6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 8</td>
<td>timestamp</td>
<td>11520</td>
<td>pkt len = 6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 9</td>
<td>timestamp</td>
<td>14080</td>
<td>pkt len = 6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 10</td>
<td>timestamp</td>
<td>16640</td>
<td>pkt len = 6 SID</td>
<td></td>
</tr>
<tr>
<td>Seq num 11</td>
<td>ooo 11</td>
<td>timestamp</td>
<td>18560</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 12</td>
<td>ooo 12</td>
<td>timestamp</td>
<td>19520</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 13</td>
<td>ooo 13</td>
<td>timestamp</td>
<td>18240</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 14</td>
<td>ooo 14</td>
<td>timestamp</td>
<td>18880</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 15</td>
<td>ooo 15</td>
<td>timestamp</td>
<td>19200</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 16</td>
<td>ooo 16</td>
<td>timestamp</td>
<td>20480</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 17</td>
<td>ooo 17</td>
<td>timestamp</td>
<td>20800</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 18</td>
<td>ooo 18</td>
<td>timestamp</td>
<td>19840</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 19</td>
<td>ooo 19</td>
<td>timestamp</td>
<td>21440</td>
<td>pkt len = 6 SID</td>
</tr>
<tr>
<td>Seq num 20</td>
<td>ooo 20</td>
<td>timestamp</td>
<td>23680</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 21</td>
<td>ooo 21</td>
<td>timestamp</td>
<td>24000</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 22</td>
<td>ooo 22</td>
<td>timestamp</td>
<td>24320</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 23</td>
<td>ooo 23</td>
<td>timestamp</td>
<td>24960</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 24</td>
<td>ooo 24</td>
<td>timestamp</td>
<td>25280</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 25</td>
<td>ooo 25</td>
<td>timestamp</td>
<td>26240</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 26</td>
<td>ooo 26</td>
<td>timestamp</td>
<td>26650</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 27</td>
<td>ooo 27</td>
<td>timestamp</td>
<td>27200</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 28</td>
<td>ooo 28</td>
<td>timestamp</td>
<td>27700</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 29</td>
<td>ooo 29</td>
<td>timestamp</td>
<td>29120</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 30</td>
<td>ooo 30</td>
<td>timestamp</td>
<td>29760</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 31</td>
<td>ooo 31</td>
<td>timestamp</td>
<td>30400</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 32</td>
<td>ooo 32</td>
<td>timestamp</td>
<td>31040</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 33</td>
<td>ooo 33</td>
<td>timestamp</td>
<td>32360</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 34</td>
<td>ooo 34</td>
<td>timestamp</td>
<td>32960</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 35</td>
<td>ooo 35</td>
<td>timestamp</td>
<td>33280</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 36</td>
<td>ooo 36</td>
<td>timestamp</td>
<td>33920</td>
<td>pkt len = 61</td>
</tr>
<tr>
<td>Seq num 37</td>
<td>ooo 37</td>
<td>timestamp</td>
<td>34560</td>
<td>pkt len = 61</td>
</tr>
</tbody>
</table>
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 = 2240, 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 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 38 ooo 43      timestamp = 31680, pkt len = 61
Seq num 40 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 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-hdrext:csrc-audio-level
  a=rtp-hash:1.10 1c812535e27b518413c4146a20fd5e715704da9c591ae32d58ee6fed6d40f
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=rtp-hash:1.10 c1a98e15f12937b9cad2488c6091468f7610eefefa59863c77827669b913f38

00:00:00.058.644 INFO: DSPfindDerStream() found HI interception point ID 10g-devl, 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, 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
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

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

HCODEC DSCodecCreate( void* pCodecInfo, unsigned int uFlags ); /* if DS_CC_USE_TERMININFO 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 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** \(^1\) 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 ±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
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

---

Group Call
- Unrecognized / low confidence speech
- Using GPUs when possible
- Group Call
  - ASR Text, data
  - RTP audio
  - SigSRF Software
  - Kaldi Run-Time Libs
  - Telco 5G Network
  - KubeEdge ASR Offloading

© Signalogic 2020
Rev 1, Aug 2020
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