Weather Station System Design

Kry10

Goals and Motivation

- Showcase **KOS libraries** and **core apps**
- Demonstrate **Elixir** and **Scenic** in KOS systems
- Illustrate tools for **building and deploying** KOS systems

System Overview

Runs on:

- BeagleBone Black (BBB)
- Sensor
  - **BME280 sensor**
  - **ARM QEMU simulator** (fake sensor)

Uses **KOS Poukai manifest** + 3 user-level apps
- Weather sensor - Weather station - I2C

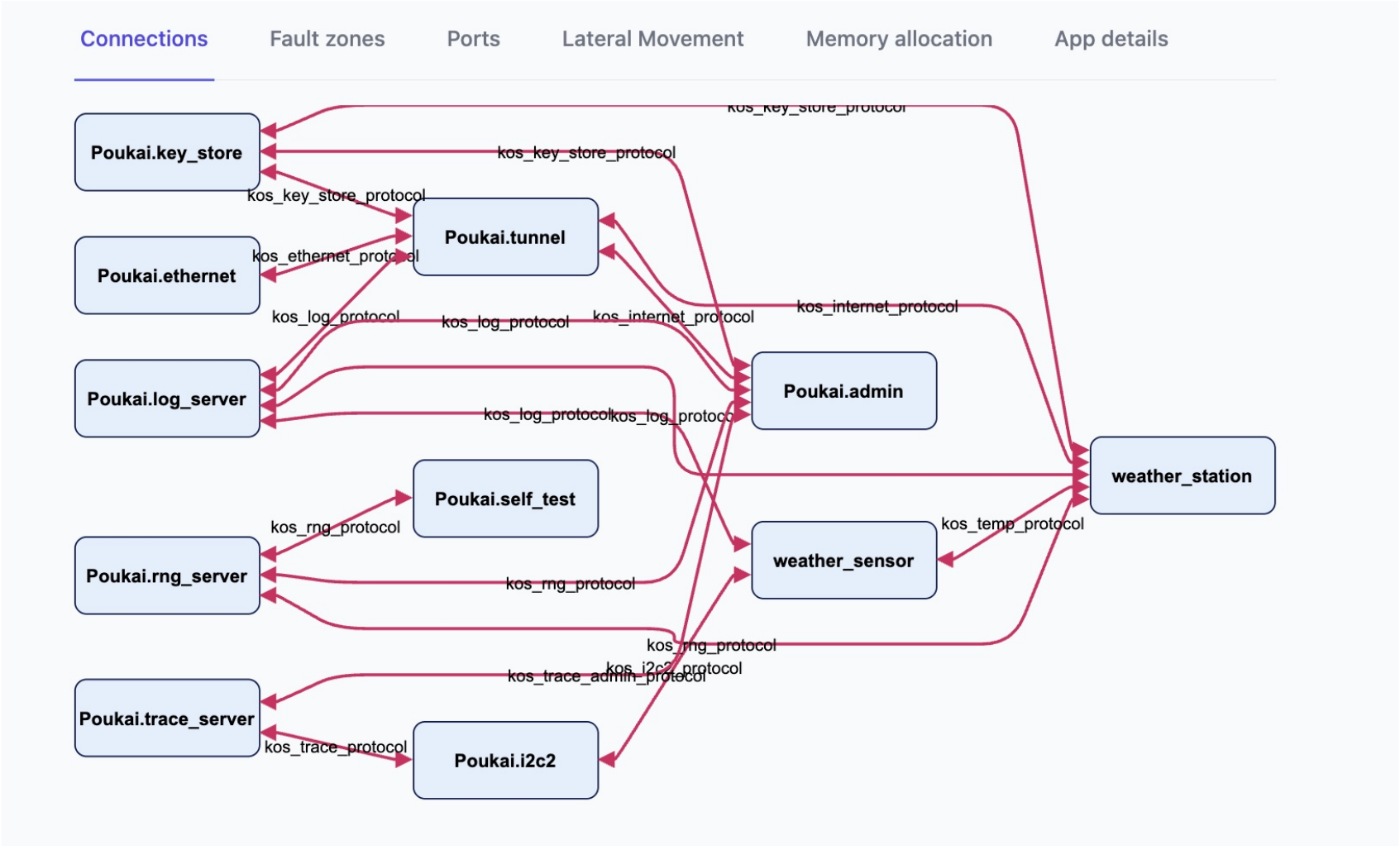

Figure 1: Weather Station

Core KOS Apps

Pre-installed Apps:

- `self_test`: performs a series of tests on startup
- `admin`: Erlang BEAM VM
  - coordinates admin activities (restart, upgrade),
  - client to a Kry10 Server
- `key_store`: a way to store secrets, private keys, that aps can get at runtime
- `ethernet`: ethernet driver and implements the ethernet protocol to provide access to apps
- `tunnel`: secure communication to host machine (typically Studio server)
- `rng_server`: random number generator for cryptographic security
- `log_server`: system-wide logging*

User-Level Apps

Custom Apps:

- `weather_sensor`: Reads **BME280 sensor**, controls **LED**
- `i2c`: **I2C driver** for BeagleBone Black (not on QEMU)
- `weather_station`: **GUI app** using **Scenic** (Elixir framework)

`weather_sensor` App

Provides sensor data via **weather protocol**
Uses:

- **BME280 driver** (BBB)
- **Fake driver** (QEMU)
- **I2C Server** for **sensor communication**
- **GPIO control** for **LED**

BME280 Driver

- Location: `weather_sensor/src/am335x`
- Configuration: `resources.dti`

Communicates via I2C

- Reads temperature, humidity, pressure
- Uses **GPIO** to control LED

QEMU Virtual Sensor

Location: `weather_sensor/src/qemu-arm-virt`

Emulated sensor:

- Returns **static values** instead of real readings
- Simulates LED **on/off messages**
- **Does not use I2C**

Weather Protocol

Defined in `weather_protocol/weather_protocol.h`

Supported Messages:

1. **Read sensor data**
2. **Turn LED ON**
3. **Turn LED OFF**

Scenic

Scenic is an application framework written directly on the Elixir/Erlang/OTP stack. With it, you can build client-side applications that operate identically across all supported operating systems, including MacOS, Ubuntu, Nerves/Linux, and more.

Scenic is primarily aimed at fixed screen connected devices (IoT), but can also be used to build portable applications.

https://hexdocs.pm/scenic/welcome.html

Scenic GUI (Elixir)

Elixir-based UI for displaying sensor data
Built using Scenic Framework

Communicates with:

- `weather_sensor` (retrieves sensor data)
- **Device LED control**
- **Remote GUI via Kry10 Server**

Admin Server & Kry10 Server

Admin server handles:

- **Restart, upgrade operations**
- **WebSocket-based CNC commands** (from Kry10 Server)

Kry10 Server Authentication

- Uses **public/private keypair**
- Secure **challenge-response** handshake

Remote Management

Kry10 Server:

- Manages **connected KOS devices**
- Provides **web-based remote access**
- **Secure tunnels GUI traffic**

Client functions:

- Restart (`system_restart`)
- Upgrade (`system_upgrade` via URL)

Secure Tunnel & Ethernet

Uses **BoringTun** (WireGuard® VPN)
Enables **encrypted off-device communication**

Network Flow:

1. `tunnel` app → **encrypts data**
2. `ethernet` server → **routes data**
3. Kry10 Server **decrypts & relays**

Summary

- **KOS system design** for IoT & embedded
- Uses **Elixir, Scenic, I2C, GPIO, TLS tunnels**
- **Secured admin access** via **Kry10 Server**