Sick sensingCam

The SEC110-5C9D1SFZZZZ (part no. 1144993) is the flagship variant of SICK's **sensingCam SEC100** series—essentially a ruggedized, self-contained 5 MP IP video sensor that exposes both **RTSP/MJPEG streams and a fully documented REST API**.

Below is an overview and integration concept to bring the sensingCam into an existing open-source edge stack (e.g., **MING = MQTT + InfluxDB + Node-RED + Grafana**) while considering loss-less video capture via an OSS NVR such as Shinobi or Frigate. Integration in this manner will enable further capabilities and use cases for the sensingCam available via a decoupled controls architecture.

This document is conceptual based on preliminary review; some inaccuracies may be present.

1. Electro-optical and mechanical highlights

Feature	Detail	Notes
Sensor	5 MP CMOS, 2 880 × 1 616 px, rolling shutter	Raw resolution decides FFmpeg / GStreamer pipeline bandwidth.
FOV	82° × 52°, WD 300 mm – 10 m	
Frame-rate	5–30 fps; 25 fps full-res; 30 fps @ 1080p/720p	Tune detector FPS in Frigate/Shinobi.
Video codecs	H.264, H.265, Motion-JPEG; two concurrent RTSP + one MJPEG stream	
On-board buffer	10 s @ 5 MP → 40 s @ 720p before + after trigger (MP4)	
Interfaces	100 Base-TX, REST API, 1× digital IN, 12–24 V DC, IP65, M12 connectors	
Power	2 W typ., 10 W peak	

2. Firmware services & API surface

2.1 Live streaming

- **RTSP endpoints** (configurable paths /stream1 | /stream2) negotiated via SDP, CBR/VBR selectable; H.264 default @ 8 Mb s^{-1} (support.sick.com).
- **MJPEG over HTTP** for browser-native dashboards (SICK).

2.2 Snapshot & event recording

Both are first-class REST resources; they can be triggered by the digital input, the web-UI or an HTTP POST /api/v1/trigger call. Tutorials confirm identical auth flows for snapshots and events (support.sick.com, support.sick.com).

2.3 REST API structure

- Swagger-like **deviceDescription.yaml** is downloadable from http://<ip>/deviceDescription.yaml or from the web-UI since FW 2.0.2 (support.sick.com).
- Auth: SHA-256 digest; default credentials main | servicelevel (change immediately) (support.sick.com).
- SICK publishes **Postman/Insomnia collections** and a Python demo that wraps the API, simplifying CI/CD integration (support.sick.com).

3. Open-source video capture layer

3.1 Why an external NVR?

The camera's 6 GB ring buffer is excellent for short event clips, but continuous archiving or AI inference consumes orders of magnitude more storage/compute. OSS recorders let you scale independently.

OSS NVR	Integration notes
Shinobi (Node.js) – native RTSP/MJPEG, REST & WebSocket API, event hooks for MQTT	Same language as Node-RED; easy to call Shinobi's /api/recording/{mid} to push clip metadata into Influx.
Frigate (Go + FFmpeg) – Docker-only; edge-TPU/GPU acceleration; publishes rich MQTT JSON for each object event	Drop-in for MING because MQTT topics already carry timestamps and file paths.
ZoneMinder (C++) – mature, but heavier ; RTSP tested on RPi 4	Good if you need ONVIF discovery.

A minimal **docker-compose** excerpt for Frigate + Mosquitto looks like:

```
services:
 MQTT:
  image: eclipse-mosquitto:2
  ports: ["1883:1883"]
 frigate:
  image: ghcr.io/blakeblackshear/frigate:stable
  shm_size: "512m"
  ports: ["8971:8971","8554:8554"]
  volumes:
   - ./frigate.yml:/config/config.yml
   - ./media:/media/frigate
  environment:
   FRIGATE_RTSP_PASSWORD: "${RTSP_PASS}"
```

(The clips path written by Frigate becomes the **source-of-truth URI** that we will correlate against IoT events.)

4. Building the MING pipeline

The MING stack (MQTT → InfluxDB → Node-RED → Grafana) is increasingly the de-facto IIoT historian pattern (InfluxData, Prescient Devices, balena Blog).

4.1 Data plane

```
[PLC / sensor] – MQTT –► Node-RED –► InfluxDB
                              |
                              | REST/JSON
                              ▼
                 SICK sensingCam (trigger)
```

1 . **Node-RED** flow subscribes to the same Mosquitto broker topics that drive machine events. When an 'anomaly' flag arrives, it:

- calls POST /api/v1/event/recording/start on the camera (or toggles digital IN via GPIO) (support.sick.com);
- waits for the camera to push the MP4 file to its FTP target (or queries Shinobi/Frigate REST once the clip is complete);
- writes a time-series point:

```
{
  "measurement": "machine_events",
  "tags":   { "line":"#3", "camera":"SEC110", "event":"stop" },
  "fields": { "video":"/nvr/clips/2025-07-24/stop_12-03-02.mp4" },
  "time":   2025-07-24T12:03:02Z
}
```

(The video field stores a relative path; Grafana Explorer can render it via a **Discrete or Table panel** with an HTML cell plug-in.)

Ready-made Node-RED nodes (node-red-rtsp-to-mjpeg, node-red-contrib-ffmpeg-iotcam) streamline RTSP ingestion and frame grabs; example dashboard throttling strategies are documented in the Node-RED forum.

4.2 Visualization

- **Grafana** connects to InfluxDB (Flux queries) and overlays machine metrics with clip hyperlinks; "live-mode" panels render near-real-time MQTT messages side-by-side with archived MP4s.
- Because Shinobi and Frigate each expose unauthenticated **HLS / WebRTC restreams**, Grafana's iframe panel can show the camera live while its timeline tracks IoT KPIs.

5. Performance & bandwidth tuning

SICK's own guidance emphasizes codec choice, bitrate and FPS as primary levers:

Lever	Heuristic
H.265 > H.264 > MJPEG	Up to 50 % bandwidth savings at equal PPS.
Bitrate	8 Mb s^{-1} default fits 5 MP @ 25 fps; halve when dropping to 1080p.
FPS	8 fps is a sweet-spot for ML inference on Jetson Nano; drive digital IN at full 30 fps only during debug.
GStreamer pipeline	queue ! videoconvert ! video/x-raw,framerate=10/1 before encode prevents buffer bloat.
Disk I/O	Map Frigate /tmp/cache to tmpfs to spare SSD wear.

6. Security hardening

- Change default passwords and disable unused API endpoints (ACL in deviceDescription.yaml).
- Place RTSP/MQTT on an isolated VLAN; expose Grafana via HTTPS reverse proxy only.
- Use MQTT TLS + auth; Mosquitto allows per-topic ACL so the camera can publish but not subscribe.
- Camera firmware ships signed; validate SHA-256 before OTA.

jstearns@murrinc.com

7. Developer take-aways

1. **API-first** – The REST interface is rich enough to script every operation; SICK even ships a Python wrapper and Postman collections.
2. **Edge-side pre-buffer** – Let the camera's 40 s circular buffer handle pre-event evidence; trigger retrieval via Node-RED to avoid recording idle time.
3. **Dual-stream architecture** – Feed a lightweight 720p/H.264 stream to analytics, reserve the native 5 MP stream for archival.
4. **Correlate by timestamp**, not filenames. InfluxDB's nanosecond precision easily aligns PLC cycle-times with video frame times.
5. **Hardware decoding matters** – A Raspberry Pi 4 can push 6× 1080p RTSP streams if GPU memory is bumped, but only ~3 streams once Node-RED starts transcoding.
6. **MING is glue, not glass** – Keep Grafana read-only; let Node-RED orchestrate side effects (API calls, MQTT publications).

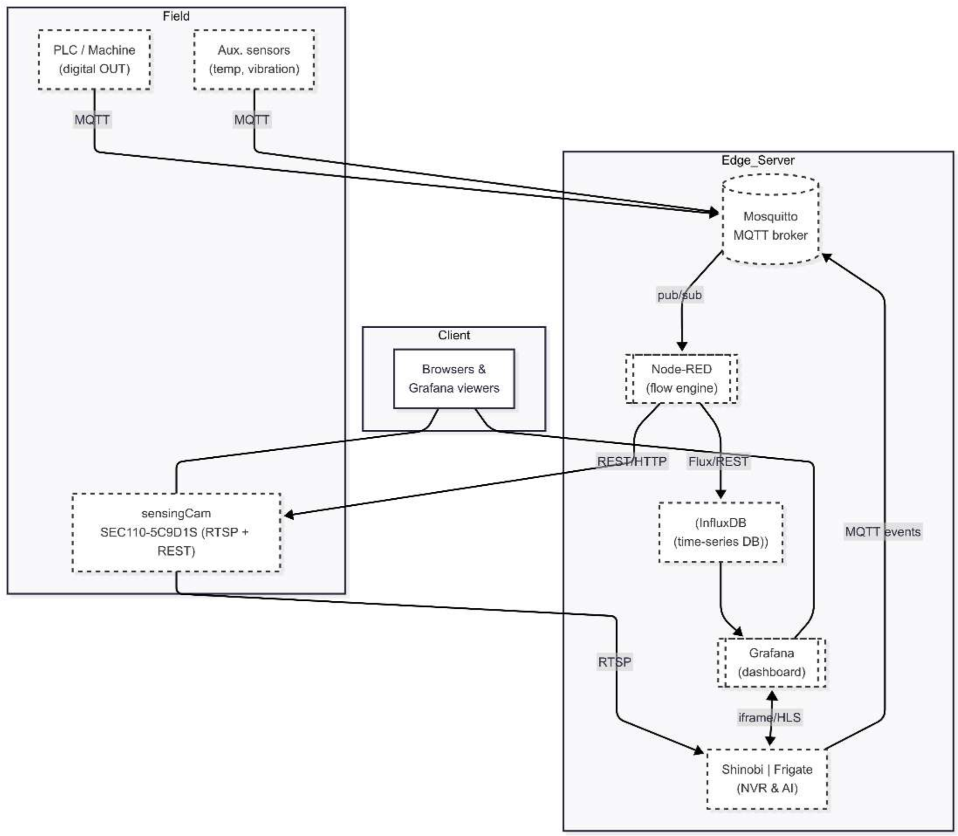
8. Further resources

- https://www.sick.com/media/pdf/0/20/220/dataSheet_SEC110-5C9D1SFZZZZ_1144993_en.pdf "sensingCam SEC100 SEC110-5C9D1SFZZZZ, Data sheet"
- https://www.sick.com/sg/en/catalog/products/machine-vision-and-identification/machine-vision/sensingcam-sec100/sec110-5c9d1sfzzzz/p/p683220?tab=detail "SEC110-5C9D1SFZZZZ - sensingCam SEC100 - SICK AG"
- https://support.sick.com/sick-knowledgebase/article?id=e44f7fea-ecb0-4d24-a163-573989d4792d "
- https://support.sick.com/sick-knowledgebase/article/?code=KA-09885
- https://support.sick.com/sick-knowledgebase/article/?code=KA-09886
- https://support.sick.com/sick-knowledgebase/article/?code=KA-09939 "Download REST API device description file from sensingCam SEC100"
- https://support.sick.com/sick-knowledgebase/article?id=4731c5ec-97fe-4b3c-a490-dca53d627d56 "
- https://shinobi.video/ "Shinobi"
- https://shinobi.video/features "Features - Shinobi"
- https://docs.frigate.video/frigate/installation/ "Installation | Frigate"
- https://forums.zoneminder.com/viewtopic.php?t=29681 "Raspberry Pi 4 with 8 RTSP cameras - monitor only"
- https://www.influxdata.com/blog/-ming-stack-introduction-influxdb/ "The MING Stack: What It Is and How It Works | InfluxData"
- https://www.prescientdevices.com/blog/ming-stack-guide "MING stack demystified: MQTT, InfluxDB, Node-RED and Grafana"
- https://blog.balena.io/ming-stack-mqtt-influxdb-nodered-grafana-balena/ "Accelerate IoT Development with the MING Stack - balena Blog"
- https://flows.nodered.org/node/%40bartbutenaers/node-red-rtsp-to-mjpeg "bartbutenaers/node-red-rtsp-to-mjpeg"
- https://flows.nodered.org/node/node-red-contrib-ffmpeg-iotcam "node-red-contrib-ffmpeg-iotcam"
- https://discourse.nodered.org/t/how-to-display-cctv-camera-in-dashboard-rtsp/5860?page=8 "How to display CCTV camera in dashboard (RTSP) - Page 8 - General"
- https://discourse.nodered.org/t/sending-rtsp-camera-streaming-over-internet/46606?page=3 "Sending rtsp camera streaming over internet - Node-RED Forum"
- https://community.grafana.com/t/how-to-visualize-in-live-the-data-from-influxdb/61633 "How to visualize in live the data from InfluxDB"
- https://flows.nodered.org/flow/127b038961f873d1babeecaf5578959e "RTSP Grab Frame (flow) - Node-RED"
- https://www.influxdata.com/community-showcases/node-red-influxdb-and-grafana-tutorial-on-a-raspberry-pi/ "255 Node-Red, InfluxDB, and Grafana Tutorial on a Raspberry Pi"

9. Visuals (Placeholder visuals, known gaps present)

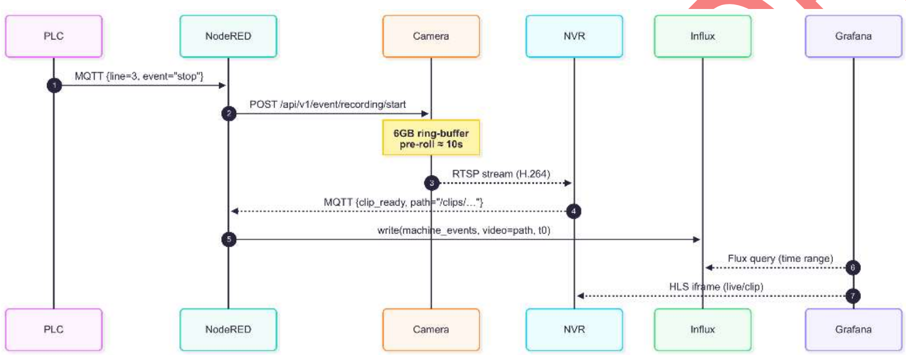
9.1 End-to-end Architecture (logical view)

```
%%{ init: { "flowchart": { "htmlLabels": false } } }%%
graph TD
  subgraph Field
    PLC["PLC / Machine<br>(digital OUT)"]
    Sensor["Aux. sensors<br>(temp, vibration)"]
    Camera["sensingCam<br>SEC110-5C9D1S (RTSP + REST)"]
  end
  subgraph Edge_Server
    Mosquitto[(Mosquitto<br>MQTT broker)]
    NodeRED[["Node-RED<br>(flow engine)"]]
    Influx["(InfluxDB<br>(time-series DB))"]
    Grafana[["Grafana<br>(dashboard)"]]
    NVR["Shinobi | Frigate<br>(NVR & AI)"]
  end
  subgraph Client
    WebUI["Browsers &<br>Grafana viewers"]
  end
  %% Transport
  PLC        -- MQTT          --> Mosquitto
  Sensor     -- MQTT          --> Mosquitto
  Mosquitto  -- pub/sub       --> NodeRED
  NodeRED    -- Flux/REST     --> Influx
  Influx     --> Grafana
  NodeRED    -- REST/HTTP     --> Camera
  Camera     -- RTSP          --> NVR
  NVR        -- MQTT events   --> Mosquitto
  Grafana    <-- iframe/HLS --> NVR
  WebUI      --- Grafana
  WebUI      --- Camera
  %% Footnotes (dashed style)
  classDef proto fill:#fff,stroke:#333,stroke-dasharray:5 5,color:#333;
  class PLC,Sensor,Camera proto;
  class Mosquitto,NodeRED,Influx,Grafana,NVR proto;
```


Field
PLC / Machine
(digital OUT)
Aux. sensors
(temp, vibration)
MQTT
MQTT
Edge_Server
Mosquitto
MQTT broker
pub/sub
Client
Browsers &
Grafana viewers
Node-RED
(flow engine)
REST/HTTP
Flux/REST
sensingCam
SEC110-5C9D1S (RTSP +
REST)
(InfluxDB
(time-series DB))
MQTT events
Grafana
(dashboard)
RTSP
iframe/HLS
Shinobi | Frigate
(NVR & AI)

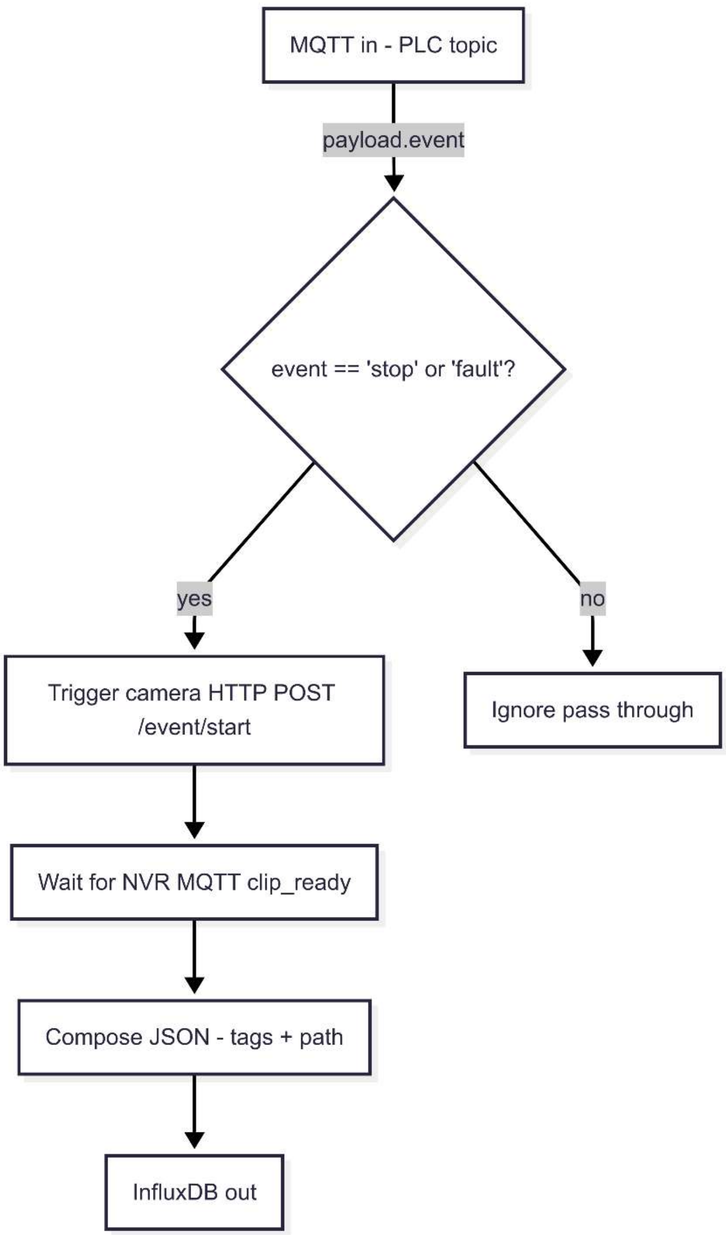
9.2 Event-centric Timeline (sequence diagram)

```
sequenceDiagram
  autonumber
  participant PLC
  participant NodeRED
  participant Camera
  participant NVR
  participant Influx
  participant Grafana
  PLC->>NodeRED: MQTT {line=3, event="stop"}
  NodeRED->>Camera: POST /api/v1/event/recording/start
  Note over Camera: 6GB ring-buffer<br/>pre-roll ≈ 10s
  Camera-->>NVR: RTSP stream (H.264)
  NVR-->>NodeRED: MQTT {clip_ready, path="/clips/…"}
  NodeRED->>Influx: write(machine_events, video=path, t0)
  Grafana-->>Influx: Flux query (time range)
  Grafana-->>NVR: HLS iframe (live/clip)
```



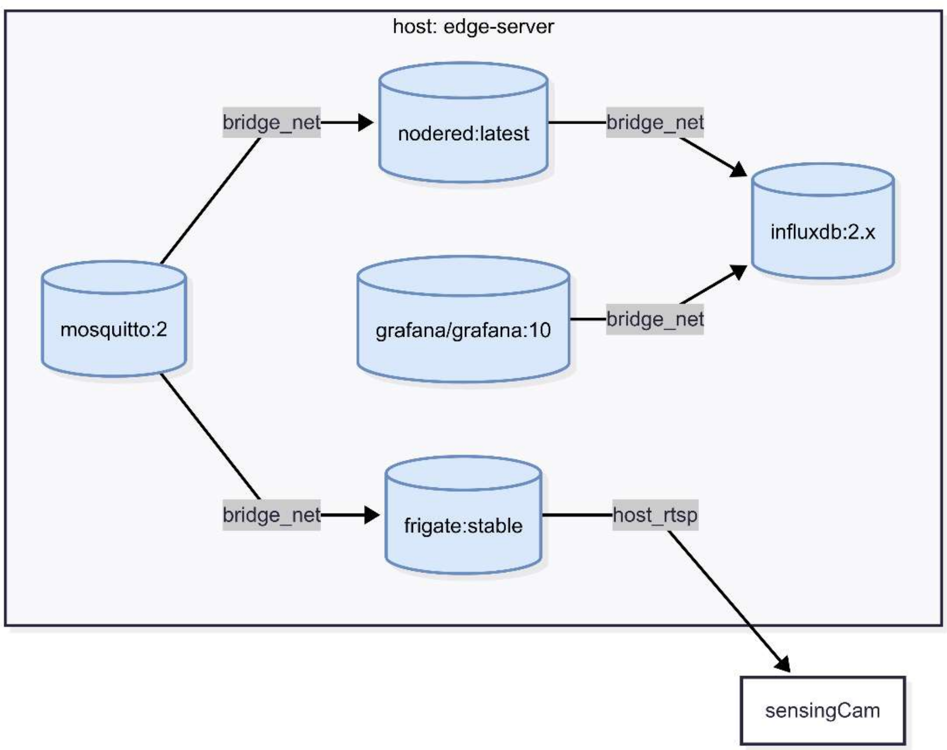
9.3 Node-RED Flow Logic (flowchart)

```
flowchart TD
  A[MQTT in - PLC topic] -->|payload.event| B{"event == 'stop' or 'fault'?"}
  B -- yes --> C[Trigger camera HTTP POST /event/start]
  C --> D[Wait for NVR MQTT clip_ready]
  D --> E[Compose JSON - tags + path]
  E --> F[InfluxDB out]
  B -- no --> Z[Ignore pass through]
```



9.4 Container-level Deployment (Docker Compose)

```
graph LR
  subgraph host: edge-server
    direction TB
    mosquitto[(mosquitto:2)]
    nodered[(nodered:latest)]
    influx[(influxdb:2.x)]
    grafana[(grafana/grafana:10)]
    frigate[(frigate:stable)]
  end
  %% networks
  mosquitto -- bridge_net --> nodered
  mosquitto -- bridge_net --> frigate
  nodered -- bridge_net --> influx
  grafana -- bridge_net --> influx
  frigate -- host_rtsp --> Camera["sensingCam"]
  classDef docker fill:#dae8fc,stroke:#6c8ebf,color:#000
  class mosquitto,nodered,influx,grafana,frigate docker
```

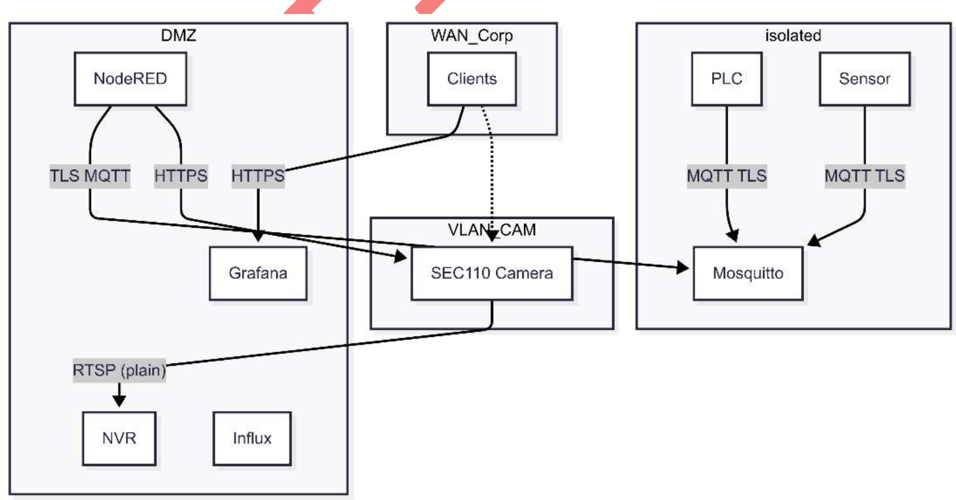


Legend – bridge_net: Docker bridge network; host_rtsp: physical VLAN that carries RTSP/HLS.

9.5 Security Zones & Trust Boundaries

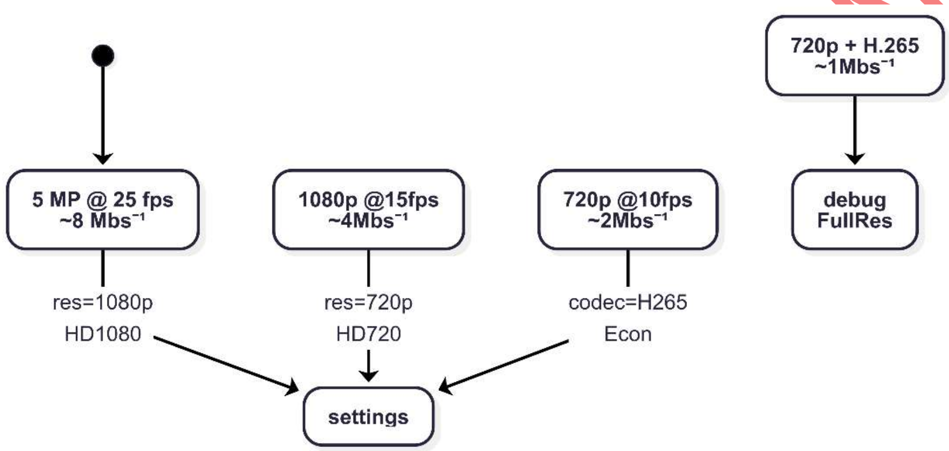
```
graph TD
  subgraph VLAN_CAM
    Camera[SEC110 Camera]
  end
  subgraph VLAN_IOT [isolated]
    PLC
    Sensor
    Mosquitto
  end
  subgraph VLAN_APP [DMZ]
    NodeRED
    NVR
    Influx
    Grafana
  end
  subgraph WAN_Corp
    Clients
  end
  PLC -- MQTT TLS --> Mosquitto
  Sensor -- MQTT TLS --> Mosquitto
  NodeRED -- TLS MQTT --> Mosquitto
  Camera -- RTSP (plain) --> NVR
  NodeRED -- HTTPS --> Camera
  Clients -- HTTPS --> Grafana
  Clients -.-> Camera
```

Dashed red line marks forbidden direct access from external clients to the camera. (Not rendered in PNG)



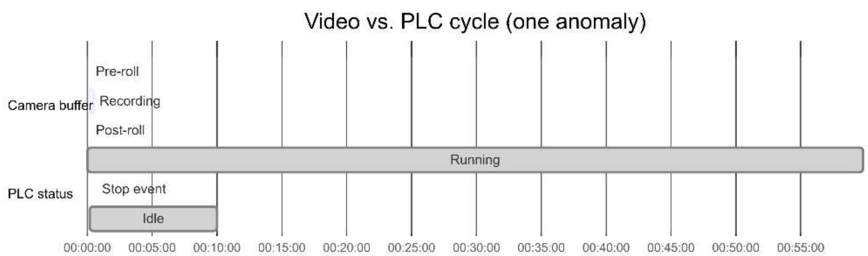
9.6 Bandwidth Tuning Matrix (state diagram)

```
stateDiagram-v2
  [*] --> FullRes
  FullRes: 5 MP @ 25 fps<br>~8 Mbs⁻¹
  FullRes -->settings: res=1080p<br>HD1080
  HD1080: 1080p @15fps<br>~4Mbs⁻¹
  HD1080 -->settings: res=720p<br>HD720
  HD720: 720p @10fps<br>~2Mbs⁻¹
  HD720 -->settings: codec=H265<br>Econ
  Econ: 720p + H.265<br>~1Mbs⁻¹
  Econ -->debug<br>FullRes
```



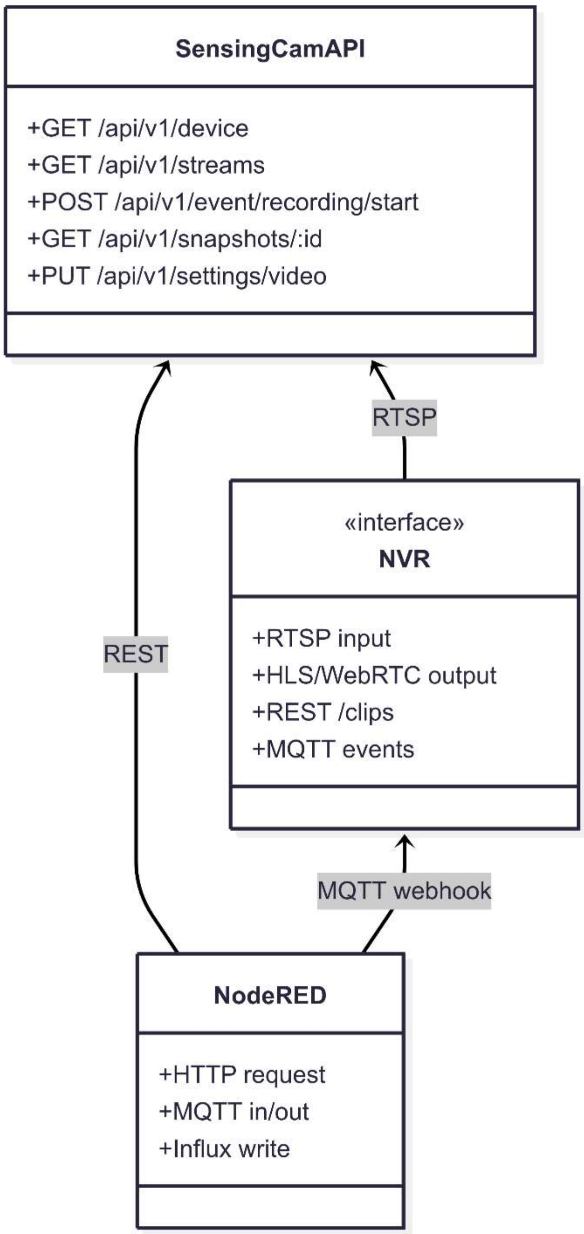
9.7 Correlation Timeline (Gantt-style)

```
gantt
  dateFormat  HH:mm:ss
  axisFormat %H:%M:%S
  title Video vs. PLC cycle (one anomaly)
  section Camera buffer
  Pre-roll             :active, pr, 12:59:50, 00:00:10
  Recording            :active, rec, after pr, 00:00:30
  Post-roll            :active, po, after rec, 00:00:10
  section PLC status
  Running              :done, 00:00:00, 00:59:50
  Stop event           :crit, 12:59:50, 00:00:40
  Idle                 :done, after po, 00:10:00
```



9.8 API Surface (class diagram excerpt)

```
classDiagram
  class SensingCamAPI {
    +GET /api/v1/device
    +GET /api/v1/streams
    +POST /api/v1/event/recording/start
    +GET /api/v1/snapshots/:id
    +PUT /api/v1/settings/video
  }
  class NodeRED {
    +HTTP request
    +MQTT in/out
    +Influx write
  }
  class NVR {
    <<interface>>
    +RTSP input
    +HLS/WebRTC output
    +REST /clips
    +MQTT events
  }
  SensingCamAPI <-- NodeRED : REST
  SensingCamAPI <-- NVR : RTSP
  NVR <-- NodeRED : MQTT webhook
```


SensingCamAPI
+GET /api/v1/device
+GET /api/v1/streams
+POST /api/v1/event/recording/start
+GET /api/v1/snapshots/:id
+PUT /api/v1/settings/video
RTSP
REST
«interface»
NVR
+RTSP input
+HLS/WebRTC output
+REST /clips
+MQTT events
MQTT webhook
NodeRED
+HTTP request
+MQTT in/out
+Influx write