RAK™
IoT Made Easy

Beyond the Dashboard:

AI-Assisted Observability for LoRaWAN Gateways

Finding the cause, not the symptom, at fleet scale

Xose Pérez @ RAKwireless

RAK™
IoT Made Easy

03:14

A gateway stops reporting.

The dashboard shows it is down.

It does not show that memory has been climbing since 22:00.

Someone drives out. The gateway reboots fine. Nothing is learned.

We monitor gateways. We do not diagnose them.

RAK™
IoT Made Easy

Why gateways resist normal observability

The constraints are physical, not architectural

1 Unattended and remote

The nearest engineer may be two hours away. Diagnosis has to happen before dispatch.

2 Constrained backhaul

Cellular links are metered. You cannot ship every log line to the cloud.

3 Thin telemetry budget

Limited CPU and flash. Instrumentation competes with the packet forwarder.

4 Fleet scale

Thousands of units, one operator. Per-device attention does not scale.

5 Failures are physical

Power, temperature, antenna, RF environment. Not only software.

APM tooling assumes a datacenter. A gateway on a pole is not one.

RAK™
IoT Made Easy

The signals already exist

Nothing here requires new instrumentation

Logs
errors, reboot reasons, reconnect events

Metrics
CPU, memory, temperature, voltage

Connectivity
RSSI, RSRP, packet loss

System events
service restarts, configuration changes

Context
firmware version, GPS, weather

What is missing

Every one of these is already collected.

They are read separately, by different people, after the incident is over.

The missing layer is correlation, not collection.

Same pattern as with other network solutions like HPE Juniper Mist or Cisco Meraki MX but for LoRaWAN.

We are not short of data. We are short of a timeline.

From anomaly to explanation

Four stages, each independently useful

1 Detect

Deviation from this gateway's own baseline, not a global threshold.

2 Correlate

Align logs, metrics and system events on a single timeline.

3 Reconstruct

Order the chain. Identify the first deviation, not the largest.

4 Recommend

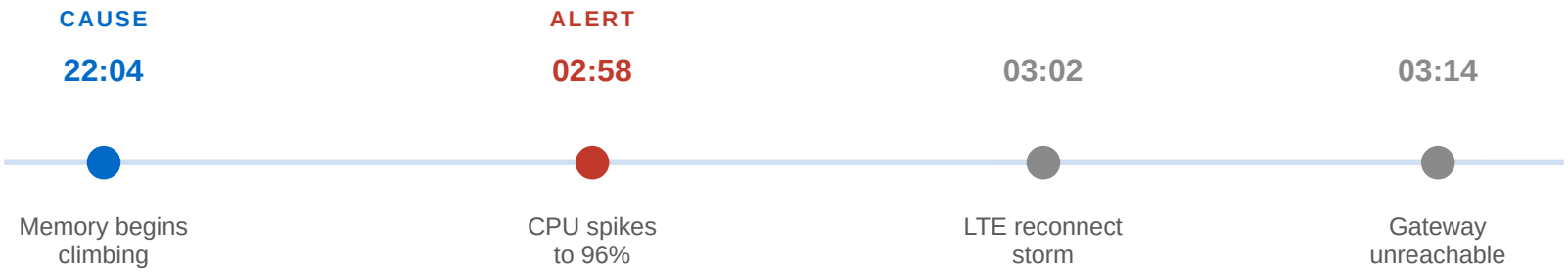
Map the pattern to a known remedy from support history.

Stage 1 alone already beats a static threshold. Stage 4 is where the value compounds.

Ship the stages in order. Each one is releasable on its own.

The loudest signal is never the first one

One incident, four events, two very different conclusions



What the threshold reports

High CPU on GW-NE-0142.
The alert fires on the largest number in the window.

What actually happened

A process leaked memory for five hours.
CPU spiked because the OOM killer was thrashing.

CPU was a consequence. We alerted on the symptom that shouts loudest.

Ordering is the signal

What changes when you model sequence instead of magnitude

Threshold-based monitoring	Sequence-based reconstruction
Evaluates each metric alone	**Evaluates events in order**
Fires on magnitude	**Fires on the first deviation**
Stateless between samples	**Carries the whole incident window**
Tells you what is loud	**Tells you what came first**

Root cause is an ordering problem before it is a machine learning problem.

What the model actually needs

One failure mode at a time, learned from labelled windows

Baseline	Degradation	Recovery
1-7 days	24 hours	24 hours

one labelled incident window

~50

confirmed cases
of the failure mode

~250

normal windows
for contrast

Memory leak is not the same problem as antenna degradation. Each failure mode needs its own labelled set.

Labelling is the real cost. It takes engineers who already know what happened.

The bottleneck is labelled incidents, not model architecture.

What accuracy actually buys you

Expected performance for a first production model

70-90%

Recall

catches most real incidents

50-80%

Precision

roughly one alert in three is wrong

Measured on held-out incidents,
not on the window the model was trained on.

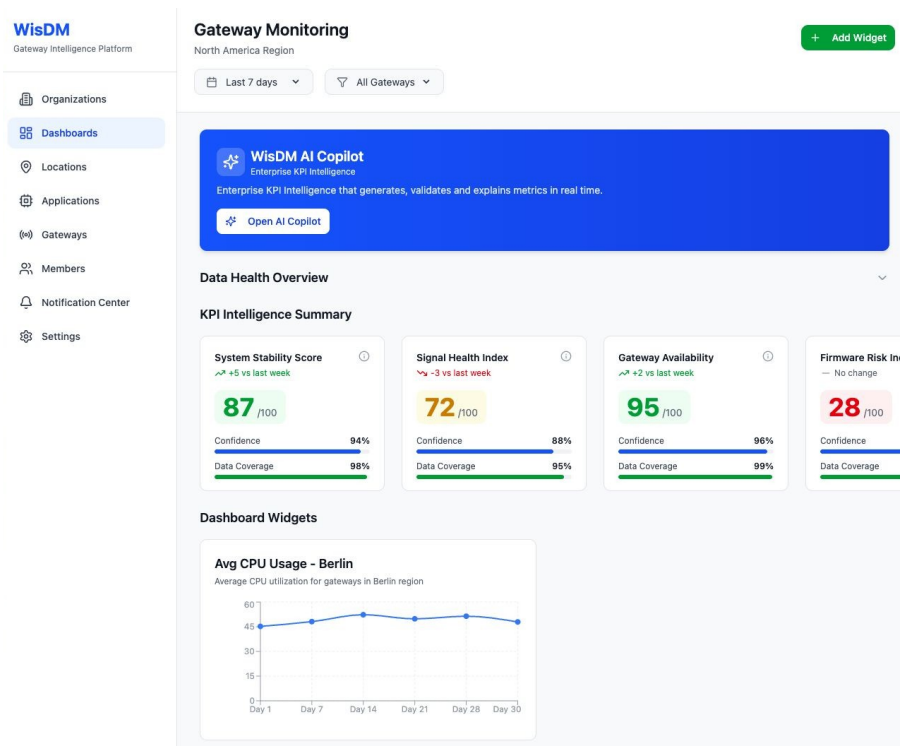
Precision decides what you are allowed to automate

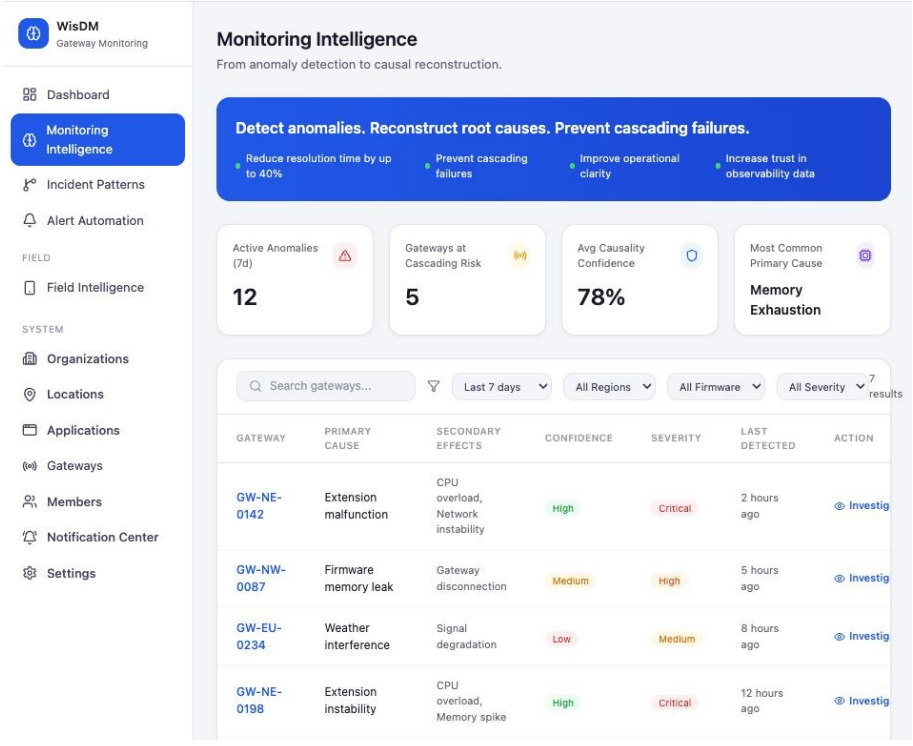
Fine	Ranking a queue an engineer was going to read anyway.
Fine	Narrowing a truck roll that was already scheduled.
Not fine	Triggering an automatic reboot on its own.
Not fine	Dispatching an engineer with no human review.

A wrong answer is cheap when a human was reading anyway, and expensive when nobody is.

How does it look like?

An insight to the actual dashboard





Meaningful information. Actionable items.

Where the analysis runs

A data-gravity decision, not only a cost decision

Continuous

Metrics and logs analysed always.

Most precise. Highest data movement over metered backhaul.

On anomaly

Metrics always, logs pulled only when something looks wrong.

Balanced. Logs move only when there is a reason to read them.

On demand

Logs analysed when an engineer asks.

Cheapest. No standing pipeline, no evidence until requested.

The binding constraint is retention. With 7 to 90 days of history, there is a hard ceiling on how much causal evidence exists at all.

This detects early. It does not predict.

- Early in the incident timeline means hours, sometimes a day.
- Not weeks ahead. That needs failure history we do not retain yet.
- Prediction becomes possible later, once labelled incidents accumulate.

Claiming prediction now would cost us the credibility that detection earns.

What changes in the field

The outcomes worth measuring

1

Time to diagnosis

From reading dashboards to reading a ranked, evidenced cause.

2

Fewer blind dispatches

Diagnose before you drive.
The two-hour trip needs a reason.

3

Reliability at scale

One operator can hold thousands of gateways in view.

We already ship gateways that adapt.

The next step is gateways that explain themselves.

RAK™
IoT Made Easy

Thank you

Xose Pérez @ RAKwireless