Store-and-forward and three-way diagnostics on the edge

WHITEPAPER · CONNECTIVITY & EDGE

Why an edge data path has to assume the network will drop — and why knowing *which* leg broke matters as much as not losing the data.

Abstract. Factory networks are not data-center networks. The link to the broker goes down, the broker restarts, a switch reboots during a shift change. The common reflex is to treat that as an exception to recover from manually — re-poll, re-stitch, hope nothing important happened during the gap. This paper argues the edge data path must instead treat connectivity loss as a normal operating condition: queue every signal locally, replay it in the order the machine produced it, and surface the health of each leg of the path separately so an operator can see exactly where it broke. Resilience and diagnosability are the same design problem.

The problem: the network *will* drop, and the gap is invisible

On a real floor, the path from a controller to a northbound system crosses equipment nobody on the OT team controls: shared switches, a plant firewall, a broker that lives in IT, sometimes a WAN link to another site. Any of those can fail for seconds or hours, and they fail at inconvenient times — a shift change, a maintenance window, a power blip.

When that happens, two things go wrong, and the second is usually worse than the first. The obvious problem is the data gap: cycles complete, alarms fire, and quality codes change during the outage, but the northbound system never hears about them. The subtler problem is *attribution*. When the operator finally notices a hole in the dashboard, the symptom has already reached production — and they have no way to tell whether the source stopped responding, the pipeline choked, or the sink simply couldn't reach the broker. A single "no data" light tells you that something is wrong, not where. That ambiguity is what turns a five-minute network blip into an afternoon of guessing.

The idea: treat the outage as a normal state, not an exception

The premise that makes an edge data path trustworthy is that connectivity loss is expected, not exceptional. Two design commitments follow from it.

First: queue at the source, replay in source order. If the path can break anywhere downstream, the only safe place to hold data is as close to the machine as possible — at the source, on the gateway itself, before anything depends on a network hop. The buffer is built to *preserve* what the machine produced: each signal is queued locally with its quality code intact, so the reading that arrives after recovery carries the same meaning it had when the controller emitted it. And it replays in the order the machine produced it — source order — so the downstream sees a faithful sequence, not an arbitrary reshuffle. There is no manual re-poll and no recovery step for an operator to remember.

Second: make each leg of the path observable on its own. "Is data flowing?" is the wrong question, because it has one answer for three very different failures. The right model separates the path into three legs — **source**, **pipeline**, and **sink** — and reports the health of each independently. Did the controller stop answering? That's a source fault. Is the transform backing up? That's a pipeline signal. Can't the path reach the broker? That's a sink fault. When the legs are surfaced separately, the operator sees *which* one broke during the outage, not just that something did — before they have to reverse-engineer it from a missing number in a report.

Why these two commitments reinforce each other

Resilience without diagnosability is a black box that happens to recover; diagnosability without resilience is a clear view of data you've already lost. Together they make the path operable.

- **Buffering converts a hard failure into a deferred delivery.** When the sink can't reach the broker, the source-side queue absorbs the gap and the sink leg goes red — a visible, bounded condition rather than silent loss. On reconnect, the queue drains in source order and the leg goes green. The outage becomes a state you watched pass, not a hole you discovered later.
- **Fanout is independent per sink, so one bad leg doesn't poison the others.** A floor often sends the same canonical stream to more than one destination. Each sink keeps its own position in the queue, so a destination that's unreachable falls behind and catches up on its own — without holding back a healthy destination. One red leg stays one red leg.
- **Three-way diagnostics tell you the blip was *only* a blip.** Because source, pipeline, and sink are separate, a clean source and pipeline with a red sink is unambiguous: the machine kept producing, the gateway kept queuing, and only delivery was deferred. That is the difference between "we lost an hour of the line" and "delivery caught up after the switch came back."

How Elpis does it

EdgeConnect implements both commitments as core runtime behavior, not as an add-on. **Store-and-forward is per-route**: every route buffers locally, and each sink on a route keeps its own cursor into that buffer — which is what lets one sink fall behind and recover without affecting another. Signals queue at the source with their quality code preserved, and on reconnect they replay in source order. The buffer is built to preserve what the machine produced through an outage; it is a designed-to-preserve design claim about ordering and quality, not a promise that nothing can ever be lost.

Three-way diagnostics are always on: source, pipeline, and sink health are surfaced separately so the operator can see which leg broke the moment it breaks — during the outage, not after the symptom reaches a report. This is observational. It reports the state of the path; it does not steer it.

The path is also **offline-first by design**. Cloud connectivity is opt-in, not required. A plant on an isolated OT VLAN — or fully air-gapped — runs the same way as a plant with internet access: the same buffering, the same source-order replay, the same three-way health view, with the gateway carrying its own per-plant identity established at first start. The locked architecture treats the per-plant runtime as the unit of resilience; multi-site visibility comes from aggregating across those runtimes, not from a shared dependency that becomes a single point of failure.

For the column model that situates the edge runtime in the wider stack, see **/architecture**; for the EdgeConnect runtime story and the protocols it collects today, see **/capabilities/connectivity-edge**.

What this is *not*

- **Not "never loses data."** The buffer is built to preserve what the machine produced and to replay it faithfully in source order — that is a guarantee about ordering and quality through an outage, not a claim that loss is physically impossible under every condition. Storage and retention are finite; the design makes the failure visible and bounded rather than silent.
- **Not a replacement for SCADA, historian, or MES.** Store-and-forward feeds your northbound systems faithfully after a gap; it doesn't take over operator HMIs, control logic, or alarm acknowledgment, and the historian stays the long-term archive of record if that's your policy. The edge path sits beside what you already run.
- **Not an AI claim.** Recovery and diagnostics are deterministic runtime behavior, not learned. Diagnostics observe the path; they don't drive it. Where AI appears in the platform, it is decision-support outside the data path.
- **Not a global-ordering promise.** Replay is faithful to the order each source produced — per-source, per-sink. It does not reconstruct one global timeline across every source on the floor; only the order that's actually meaningful is the order that's preserved.

Takeaways

1. Treat connectivity loss as a normal operating state, not an exception — the edge path should queue through it without a manual recovery step.
2. Queue at the source and replay in **source order**, with each signal's **quality code preserved**, so what arrives after a gap carries the meaning it had when the machine produced it.
3. Make **source / pipeline / sink** observable separately, so an operator sees *which* leg broke during the outage — before the symptom reaches production.
4. Resilience and diagnosability are one design problem: buffering converts silent loss into a visible, bounded, self-draining state.
5. Offline-first is structural — isolated and air-gapped sites run the same path, cloud strictly opt-in, identity per plant.
