# DIAGNOSING PARTIAL OUTAGES ACROSS CONNECTED SYSTEMS

**By: Jeff Collins**

www.wanaware.com

**Version 1.1 — January 2026**

# Executive Summary:

Partial outages are hard because the work is broken, but the tools say it isn't. People cannot log in, check out, use VPN, or complete a critical task, while dashboards show servers responding, services returning 200s, links up, CPU and memory within range, and vendor status pages marked green.

Most monitoring tools report the health of individual components, not whether a real user action completes from start to finish. When a workflow fails across identity, DNS, certificates, gateways, networks, and third-party services, teams lose time proving what is failing, where it breaks first, who is affected, and who owns that step.

The fastest way to reduce downtime is to name the failing action, trace its path, find the first place where success drops off, or reachability fails, and capture four proof points that speed ownership: first break, time window, pattern, and boundary. This turns partial outages from debate into diagnosis and helps restore service with less risk.

# When “it’s down” doesn’t mean down

When teams say “it’s down,” they usually mean the work: login, VPN access, checkout, key APIs, EHR access, call center tools, payment rails, or OT visibility. In connected environments, “down” is rarely a clean outage. More often, it is a partial outage.

A partial outage shows up as split results at the same time:

- The same action works for some people and fails for others
- One office works while another cannot
- One ISP path succeeds while another times out
- A retry works once, then fails again
- Dashboards look healthy, but the workflow does not complete

This is why these incidents drag. The failure is real, but teams cannot agree on what is failing in a way that points to an owner. If you cannot name the failure, you cannot prove where it breaks first or who owns that step.

The uncertainty is what makes it stressful. Tickets spike, leaders get pulled in, and customer-facing teams need a clear explanation. Vendors often see healthy indicators on their side as well, so they ask for specifics that narrow the problem.

Use a simple Proof Pack during incident calls. Fill in these four items and escalations move:

- **First break:** the first dependency where success rates drop
- **Time window:** start time, peak time, recovery time
- **Pattern:** who is affected and what they share (region, ISP path, identity route, endpoint)
- Boundary: what fails versus what still works

The same questions come up in every incident: what is unavailable, where does it break first, and who owns that step.

# The core idea: availability is a completed journey

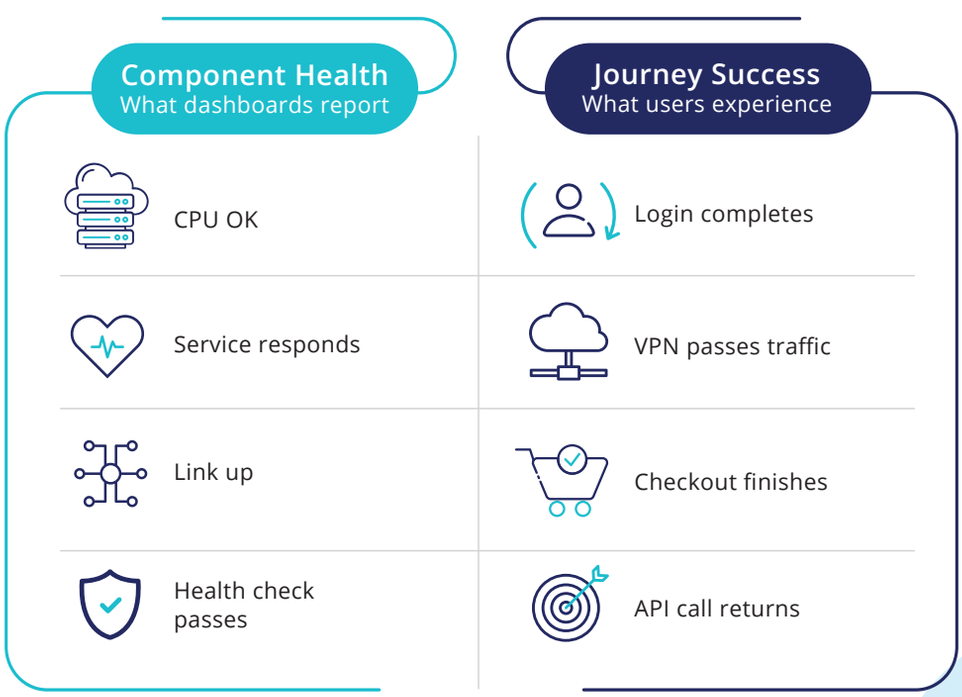
Most teams still measure availability as “Does the system respond?” Users measure it as “Can I finish the action?”

That difference matters because a login is not one system. It is a chain of steps that have to work in order: identity, DNS, certificates, gateways, the application, and the routes between them. If one step in the chain fails, the action fails, even if every individual component still looks healthy on its own dashboard.

Here is the simplest way to say it: availability breaks at the journey level, but most tools report component health. That is why partial outages are so hard to diagnose. The failure is real, but it does not show up as one broken thing you can point to and name.

The diagram below shows the difference between what dashboards report and whether the action actually completes.

## Component Health vs Journey Success




A component can look healthy while the action still fails.

# Before vs today: what tools were built for, and what changed

Many availability tools were built for a simpler kind of failure. When something went wrong, it was usually local and obvious:

- a server stopped responding
- a link dropped
- a database went offline
- a region went completely down
- an application tier crashed

In that world, up-or-down checks worked well. Component health was a reasonable stand-in for availability. Ownership was clear because one team usually owned the broken thing.

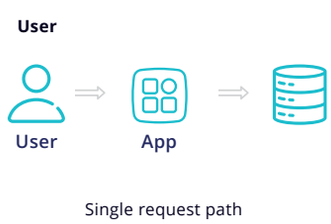
That is not how services fail today.

Now, a single business action depends on many steps working together: identity providers, DNS, certificates, gateways, shared platforms, cloud control planes, SaaS services, and third-party routes. The path is longer, and it crosses teams, vendors, and environments. A small, routine change in one place can break one step in the chain. And the failure may show up somewhere else entirely.

The diagram below shows the difference between what tools were built to see and how services actually fail now.

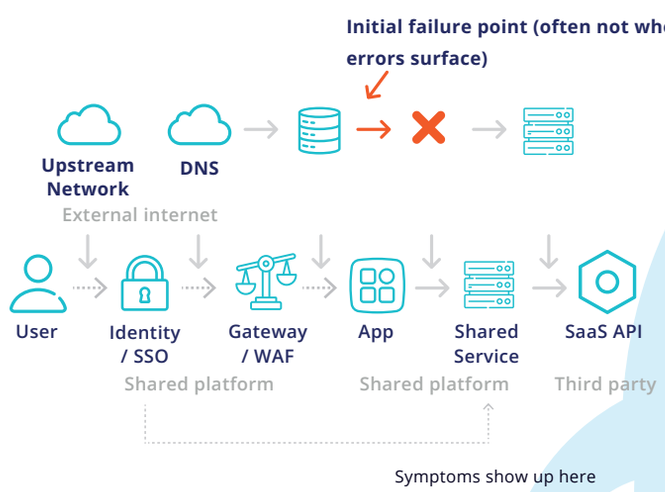
## The Monitoring Gap: When Everything Looks Healthy but the User Fails

**Then:** What tools were built for




Single environment
Binary up/down checks
Clear owner

**Now:** How services fail today




Many dependencies
Shared ownership
Symptoms surface elsewhere

Availability used to mean ‘is the system up.’ Today it means ‘can the user finish the action.

# What partial outages look like in real life

Partial outages are hard because the same workflow can take different paths for different users. The fastest way to make progress is to name the pattern you see, then confirm the boundary: what fails, what still works, and where the first break likely sits.

Below are common patterns teams report. After a few, I'll show how the Proof Pack helps you describe the failure in one shared way.

1. **Login works for some people, fails for others**
   Identity may look fine overall, but one step in the login chain breaks: token issuance, SSO, MFA, or a regional identity hop. Users get stuck in loops. Tickets spike. App logs may show 401s, but the first break often happens before the app step.
   ***Proof Pack lens:*** First break = token/SSO step. Pattern = one region or ISP path. Boundary = login fails while browsing still works.

2. **A dependency is "up," but not reachable**
   A service looks healthy on dashboards, but real traffic cannot reach it. A route change, ACL, firewall rule, or upstream gateway blocks the path. Health checks pass, yet the action fails for users.
   ***Proof Pack lens:*** First break = reachability drop at a route, ACL, or gateway. Pattern = one path/segment. Boundary = one action fails while others still work.

3. **DNS or certificate issues create phantom outages**
   Name resolution can fail on one resolver path, or certificates can fail only for certain endpoints, clients, or regions. It feels random until teams separate "can I resolve and trust the endpoint" from "is the service running."
   ***Proof Pack lens:*** First break = DNS/certificate check. Pattern = a resolver, client type, or region. Boundary = reachability fails while service metrics look normal.

4. **Availability varies by location or ISP path**
   The service works from one geography and fails from another because the route is different. Dashboards show healthy averages, but one carrier, one peering route, or one region is degraded.
   ***Proof Pack lens:*** Pattern is the giveaway: Region B + ISP X. Boundary shows what fails and what still works.

5. **A third-party issue looks like an internal outage**
   An external API, SaaS dependency, or provider degrades. The symptoms show up inside your application, so teams start in the wrong place. Progress depends on proving where the break actually starts.
   ***Proof Pack lens:*** First break = third-party edge/API. Pattern = specific endpoint/provider. Boundary = internal service is healthy, but the external call fails.

6. **Failover “works,” but the action still fails**
   Traffic shifts as designed, but the fallback relies on the same underlying dependency. The platform reports a successful failover, yet users still cannot complete the action. Teams assume recovery happened and lose time finding what stayed broken.

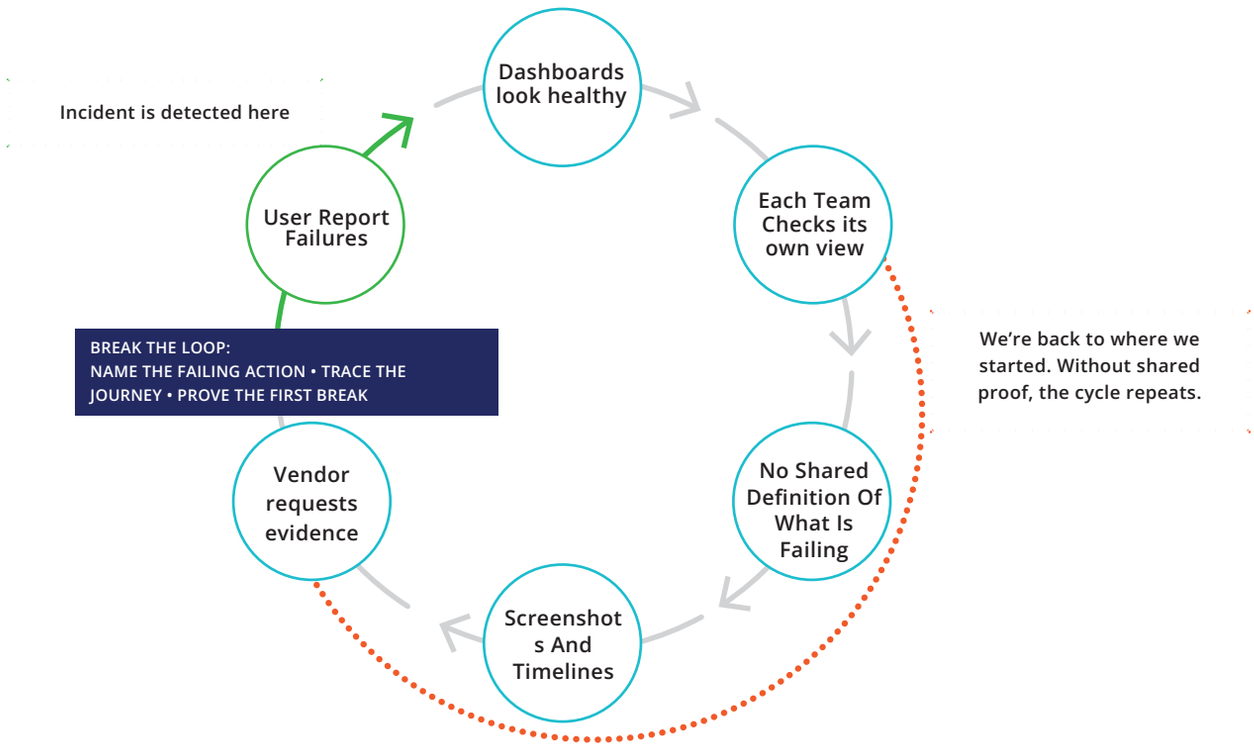
**The common thread:** once you name the pattern, you can isolate the failing step and owner faster.

## Why partial outages turn into debates

When availability breaks in pieces, the hardest part is often not the fix. It is the moment everyone realizes: we can’t point to one clear place where the action stops working. And without a shared definition of the failure, every team falls back to its own dashboard.

That is when the incident call starts to loop. The workflow is failing for real users, but the usual signals look fine. Network sees links up. Cloud sees services healthy. App teams see scattered errors. Security asks what changed. Vendors say their status is green and ask for proof. Teams trade screenshots, rebuild timelines, and keep re-checking the same views, because no one can name the failure in a way everyone can agree on.

## The Partial Outages Loop




Partial outages drag when the failure cannot be clearly named and traced.

This doesn't happen because teams are stubborn. It happens because they are looking at the problem through different windows. Each tool reports what it can see—links, hosts, services, logs—but none of them shows the full action end to end as one shared "thing" you can point to and agree on.

That is what the Proof Pack fixes. It gives everyone the same way to describe the failure: **first break, time window, pattern, and boundary.**

Once those four are clear, it gets easier to align on what's broken, who owns it, and what to do next.

Without that shared view, teams end up rebuilding the journey under pressure with screenshots, disconnected logs, and best guesses. The business waits.

# How to shorten escalations: the Proof Pack

A lot of availability work is not fixing the issue. It is **proving where the journey breaks** and **who owns that step**. The fastest escalations happen when everyone agrees on the same description of the failure.

Use the **Proof Pack** as your escalation template. **Fill in each box with one sentence.**

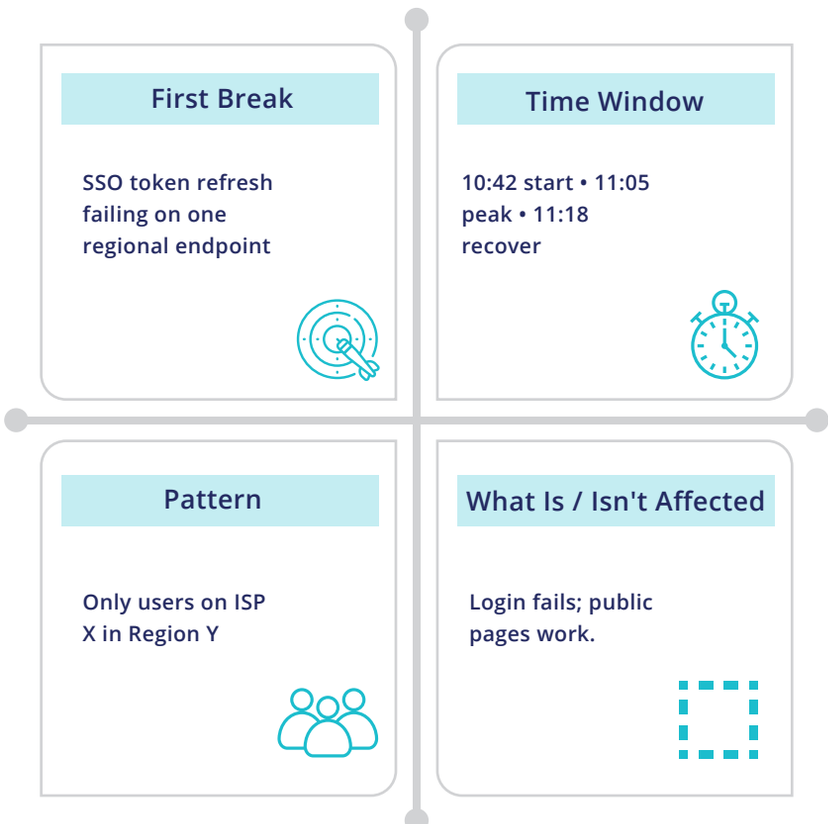
1. **First break**
   **What it is:** The first dependency where reachability drops or success rates fall.
   **Example:** "SSO token refresh failing on one regional endpoint."
   **Why it moves escalation:** It gives the conversation a starting point and an owner to engage.

2. **Time window**
   **What it is:** When it started, when it worsened, and when it recovered.
   **Example:** "10:42 start, 11:05 peak, 11:18 recover."
   **Why it moves escalation:** It narrows the search and helps teams match changes to symptoms.

3. **Pattern**
   **What it is:** Who fails, and what those users share (region, ISP path, identity route, endpoint).
   **Example:** "Only users on ISP X in Region Y."
   **Why it moves escalation:** It turns 'random' into a testable clue that others can verify.

4. **Boundary**
   **What it is:** What is affected and what is not.
   **Example:** "Login broken; public pages unaffected."
   **Why it moves escalation:** It prevents over-scoping and helps teams prioritize the right fix.

# The Proof Pack Framework for One Failing Action




When you can show these four, escalations move.

## Why this keeps getting worse

As organizations rely on a smaller set of shared platforms, a single dependency can affect many services at once. One shared control plane, identity layer, DNS path, or edge layer can interrupt multiple workflows across the business.

A practical flow that works in real incidents:

1. Name the action that is failing (login, VPN traffic, checkout, API call).
2. Pull up the journey path that supports that action.
3. Identify the first point where reachability drops or success rates fall.
4. Identify what else depends on that same dependency.
5. Apply the smallest safe change.
6. Validate recovery by re-checking the action end to end.

Do not stop at “systems look healthy.” Confirm the workflow works.

## How to evaluate an availability approach quickly

If you’re looking at a tool or process for availability, don’t start with the feature list. Start with one simple test: **when a real workflow fails, can it tell you what broke first and who needs to act?** These questions usually make that clear:

- Can it show the full path behind a failed action, like login or checkout?
- Can it point to the **first break**, not just where errors show up?
- Can it show what else depends on that same step, so you know the blast radius?
- Can it stay accurate as traffic shifts, routes change, and systems move?
- Can it produce escalation-ready proof: **first break, time window, pattern, boundary?**
- Can network, cloud, app, and security teams all trust it during an incident?

If the answer is “no” to most of these, you may still get alerts and dashboards. You will also keep rebuilding the story by hand when things get weird.

## Closing: the goal is finding the first break fast

Most availability incidents today are not clean outages. They are partial failures across a chain of dependencies. The fastest teams don't debate whose dashboard is right. They name the failing action, trace the path behind it, and find the first break.

When you can do that, you shorten escalations, avoid random changes, and get people back to work faster. The win is not just fewer outages. It's **clarity when the pressure is on**, even when everything looks "up."

Explore what service path visibility looks like for real workflows like login, VPN, and checkout.

**Explore Service Path Visibility**

*See where the workflow breaks first, not just what looks up*

www.wanaware.com

Version 1.1 — January 2026

# The method: trace the journey, find the first break, act safely

When something feels “down,” the first instinct is to hunt for a broken server or a red dashboard. In partial outages, that usually sends you in circles. The first break is often a shared step in the middle of the path and the failure shows up somewhere else.

One user action (like login or checkout) usually travels through steps like:

- Identity / authentication
- DNS
- Certificates and the TLS handshake
- Gateways, proxies, and load balancers
- Shared services (directories, databases, message queues)
- Internal networks, cloud backbones, SaaS providers, and ISP routes

Each step depends on the one before it. If one step becomes unreliable, the action becomes unreliable.

## Tracing One Failing User Action

Login or checkout

The diagram below shows what that journey looks like for one action—and why the first break is often not where users notice the problem.


Find the first failing dependency, not just the symptom.