Nick Clark · Published April 25, 2026 · PDF

How to Keep a Power Grid or Supply Chain Failure From Cascading

A single tripped substation or one stalled supplier can take down a whole network, because most systems have no governed way to bound how a local failure spreads hop to hop. This guide teaches an architectural approach for containing that spread: model the network as a credentialed topology, project each disruption through it, and emit bounded, attributed coordination directives that downstream regions can accept or refuse. The approach described here is disclosed in U.S. Provisional Application No. 64/049,409, not a shipping library, and its home is the Cascade Propagation inventive step.

What You Are Building

You are building a system that answers one question fast and defensibly: when something fails at one point in a physical network, what else is about to fail, how badly, when, and who is allowed to act on it. The failure might be a tripped substation on a power grid, a flooded port, a stalled tier-two supplier, a burst water main, or a thermal runaway in a data center. In every case the danger is the same shape: a local fault propagates through the network's connective channels and, absent any bound, keeps spreading until it has taken out regions far from the origin.

This is a real problem for grid operators, logistics planners, utility engineers, and anyone running a coordination layer over interdependent physical infrastructure. What you want is not just a bigger alarm. You want the failure to be attributed to its origin, projected forward through the actual connectivity of the network, and bounded, so response is coordinated at the specific downstream regions that are genuinely in the path, and stops where policy says it should stop.

The approach below is an architecture disclosed in a patent filing. You implement it yourself; there is no package to install.

Why the Obvious Approaches Fall Short

The standard tools each solve part of this, and the filed disclosure is candid about what they are and are not.

Domain simulators exist and work: power-grid SCADA cascade analysis, traffic simulation, epidemic modeling, supply-chain disruption modeling, structural-failure modeling. They are mature and accurate within their domain. The structural gaps the disclosure identifies are not that these tools are wrong, but that they share design assumptions that make containment hard across a real, multi-party network:

- They operate on centrally maintained models with ad hoc trust assumptions. There is no built-in notion of who is authorized to assert a given piece of the model or order a given response.
- They tend to produce unstructured alerts or a central dashboard for a human to read, rather than structured, attributable messages that downstream systems can consume and act on automatically.
- Each is scoped to a single domain. A cascade that jumps from the power domain into the transportation domain (a substation loss stranding refrigerated freight) falls between two separate models.

- When a proposed mitigation cannot or should not be applied at a downstream point, there is no first-class, recorded way for that point to say no and hand the problem back up. The rejection is silent, and the cascade continues.

So the obvious approach gives you good local physics and poor global governance. The gap is not modeling accuracy; it is the absence of a governed way to carry state and authority hop to hop.

The Architecture

The disclosed primitive treats cascade propagation as a first-class part of a governed mesh. Every element below traces to the filing.

A governance-credentialed topology graph. Represent the physical domain as nodes (substations, ports, suppliers, junctions) and edges (the propagation channels between them). The graph is not anonymous data: it is maintained by one or more governance authorities that each hold domain responsibility for their part of it. Authority over the model is explicit, so an assertion about connectivity carries the credential of whoever is responsible for it.

A per-edge propagation function. For each edge, define how a disruption at the source node projects to the connected node, with governance-policy-defined characteristics: transit, attenuation, transformation, or amplification. This is where the physics lives. An edge might attenuate a disruption (a buffer absorbs some of it), transform it (a power loss becomes a refrigeration loss), or amplify it.

A per-node aggregation function. Define how multiple incoming propagation contributions combine when they arrive at the same receiving node. A node fed by several failing upstream paths needs a rule for how those contributions add up, and that rule is policy-defined per node.

A cascade-trigger ingest interface. Disruptions enter through governed disruption observations (from the companion disruption-sensing primitive) that are mapped to the originating cascade node. This is what attributes a failure to a place in the graph.

A cascade-computation engine. This executes the propagation function across the topology and produces, per node, the predicted affected region, the magnitude, and the arrival time. That triple, where, how bad, and when, is the substance of the projection.

A cross-domain cascade composition mechanism. When a topology spans two or more domains, this combines their propagations into composite determinations, producing cascade-of-cascade results, so the grid-to-freight jump is modeled as one thing rather than lost between two models.

A cascade-authority resolution mechanism. When a topology spans multiple governance authorities, this resolves responsibility, so it is well-defined who owns a given projection or directive at a boundary.

A preemptive-mitigation directive generator. Rather than raising a dashboard alert, this produces governed coordination directives routed to the specific downstream receiving agents in the projected path.

A cascade-halting and containment mechanism. This specifies governance-policy-defined stop-conditions under which propagation is actively interrupted. This is the containment lever: policy declares where the cascade must stop, and the mechanism enforces it.

A refusal and upstream-coordination mechanism. A downstream agent that cannot or should not apply a proposed mitigation emits a refusal that is itself a first-class governed observation. Refusal reasons in the disclosure include evidential insufficiency, capability exceedance, cost-threshold, priority conflict, authority insufficiency, dispositional, safety-boundary, and composites of these. A refusal does

not cause silent failure; it lets the upstream coordinator seek an alternative mitigation, solicit corroborating observations, or escalate to a higher authority. Refusal outcomes also feed learning.

A topology-learning and adaptive-refinement mechanism. Observed propagation outcomes update the topology graph, the propagation functions, and the aggregation functions over time, per the training-governance primitive.

A cascade-lineage recording mechanism. Every topology reference, propagation computation, directive emission, mitigation, halting event, refusal, and topology update is recorded in the governance chain lineage field. This is what makes the whole thing attributable after the fact.

The unifying idea is that state and authority propagate together, hop to hop. A projection is not a free-floating number; it is credentialed, routed, refusable, and recorded.

How to Approach the Build

Work outward from the graph. The following order keeps each layer testable before the next depends on it.

1. **Model the topology as a credentialed graph.** Enumerate nodes and edges for one domain first. For each node and edge, record which authority is responsible for it. Do not start multi-domain.
2. **Write the per-edge propagation functions.** Encode transit, attenuation, transformation, and amplification per edge, driven by policy rather than hardcoded. An illustrative interface sketch, faithful to the disclosed roles and not a working library:

```
# Illustrative only. Not a shipping API.
project(edge, incoming) -> contribution:
    # returns magnitude, arrival_time, transformed_type
aggregate(node, [contribution, ...]) -> node_state:
    # policy-defined combine rule for this node
```

3. **Build the trigger ingest.** Accept governed disruption observations and map each to its originating node. Reject anything that does not carry the credential and references your policy requires.
4. **Implement the computation engine.** Traverse the graph from the origin node, applying `project` on edges and `aggregate` at nodes, emitting per-node predicted region, magnitude, and arrival time.
5. **Generate directives, do not just alert.** For each downstream node in the path, produce a governed coordination directive addressed to the responsible agent, carrying its authority credential and lineage reference.
6. **Add stop-conditions.** Define the policy predicates that halt propagation and enforce them in the traversal. Test that a cascade actually terminates where policy says it must.
7. **Make refusal first-class.** Give receiving agents a way to reject a directive with a typed reason, emit that refusal back upstream as an observation, and have the upstream coordinator branch on it (alternative, corroboration, escalation). Do not let a rejection be a dropped message.
8. **Record lineage throughout.** Every computation, directive, mitigation, halt, and refusal writes to the lineage field. Build this in from the start; retrofitting attribution is painful.
9. **Only then compose domains and add learning.** Bring in the cross-domain composition and authority-resolution mechanisms once single-domain works, and wire observed outcomes back into topology refinement last.

What This Does Not Give You

This is an architecture, not a drop-in library. There is nothing to `pip install`; you build each mechanism above yourself against your own domain.

The disclosure describes structure, not benchmarks. It states no propagation accuracy, latency, or containment-rate numbers, and this guide invents none. You must supply and validate the actual physics in your per-edge and per-node functions; the architecture governs how those functions' outputs flow, route, and are refused, but it does not certify that your grid or logistics model is correct. Garbage physics in an edge function yields a confidently attributed wrong projection.

It also does not replace domain simulators, actuation control, or forecasting. The filing positions cascade propagation as complementary to and distinct from the disruption-sensing primitive that detects and classifies the fault, the forecasting primitive that projects time-horizon predictions, and the confidence-governed execution primitive that governs actual actuation. Containment here means bounding and coordinating the response; it does not itself pull breakers or reroute trucks.

Finally, the whole approach assumes a governed mesh with credentialed authorities and lineage. If your environment has no notion of who is authorized to assert model state or order mitigations, you must establish that substrate first, or the authority, refusal, and lineage mechanisms have nothing to bind to.

Disclosure Scope

The architectural approach described in this guide, including the governance-credentialed topology graph, per-edge propagation and per-node aggregation functions, cascade computation, cross-domain composition, authority resolution, preemptive-mitigation directives, cascade-halting stop-conditions, the refusal and upstream-coordination mechanism, topology learning, and cascade-lineage recording, is disclosed in U.S. Provisional Application No. 64/049,409. This guide is educational. It is not a

warranty, a specification, or an offer of software, and nothing here should be read as a promise of performance, fitness, or availability of any product. Implementation, validation, and the correctness of any domain model you build are your responsibility.

Cascade Propagation (/cascade-propagation)

All 40 steps → (/inventive-steps)

Refusal as first-class observation. Topology that learns from every event.

Provisional application

PRIMARY TECHNICAL DISCLOSURE

- Cascade Propagation: Refusal as First-Class Observation (/articles/cascade-propagation-refusal-as-first-class-observation)

SECONDARY TECHNICAL

- Credentialed Topology Graph (/articles/cascade-propagation/credentialed-topology-graph)
- Refusal as a First-Class Governed Observation (/articles/cascade-propagation/refusal-as-observation)
- Upstream Cascade Coordination (/articles/cascade-propagation/upstream-coordination)
- Cross-Domain Cascade Composition (/articles/cascade-propagation/cross-domain-cascade-composition)
- Preemptive Cascade Mitigation (/articles/cascade-propagation/preemptive-mitigation)
- Cascade Halting Mechanisms (/articles/cascade-propagation/cascade-halting)
- Multi-Authority Cascade Resolution (/articles/cascade-propagation/multi-authority-resolution)
- Topology Learning From Operations (/articles/cascade-propagation/topology-learning)

APPLICATIONS · GENERAL

- Preventing Cross-Operator Cascade Failures in Communication Networks (/articles/cascade-propagation/communication-network-cascade)
- Preventing Power Grid Cascade Failures: Credentialed Topology and Cross-Utility Resilience (/articles/cascade-propagation/power-grid-cascade-resilience)

- Supply Chain Cascade Management (/articles/cascade-propagation/supply-chain-cascade-management)
- Coordinating IT-to-OT Cyber-Physical Cascades Across Critical-Infrastructure Sectors (/articles/cascade-propagation/cyber-physical-cascade)
- Financial System Cascade Risk Management (/articles/cascade-propagation/financial-system-cascade)
- Cross-Modal Transportation Cascade Coordination Across Aviation, Rail, Motor-Carrier, and Maritime Networks (/articles/cascade-propagation/transportation-network-cascade)
- NERC CIP Cross-Utility Cascade Containment: A Grid Cybersecurity Architecture (/articles/cascade-propagation/nerc-cip-grid)

APPLICATIONS · SPECIFIC

- GE Vernova Grid Software vs Governed Cross-Utility Cascade Propagation (/articles/cascade-propagation/ge-grid-cascade)
- Hitachi Energy Grid vs Governed Cross-Utility Cascade Propagation (/articles/cascade-propagation/hitachi-energy-grid)
- Governed Cross-Utility Cascade Handling Beyond Siemens Grid Software (/articles/cascade-propagation/siemens-grid-software)
- ABB Grid Software Alternative: Credentialed Cross-Domain Cascade Propagation (/articles/cascade-propagation/abb-grid-software)
- Emerson Ovation vs Governed Cross-System Cascade Propagation (/articles/cascade-propagation/emerson-ovation)
- Schneider Electric EcoStruxure Grid vs Governed Cross-Domain Cascade (/articles/cascade-propagation/schneider-electric-grid)
- Form Energy Alternative: Governed Cascade Propagation for Long-Duration Storage (/articles/cascade-propagation/form-energy-storage)
- Honeywell Experion vs Governed Cross-System Cascade Propagation (/articles/cascade-propagation/honeywell-experion)
- Rockwell Automation FactoryTalk vs Governed Cross-Plant Cascade (/articles/cascade-propagation/rockwell-automation)
- Yokogawa CENTUM vs Governed Cascade Propagation (/articles/cascade-propagation/yokogawa-centum)

Cascade Propagation overview → (/cascade-propagation)