Promise Networks

Why a Bilateral Link Layer Solves Congestion Control at the Source

Anjali Singhai-Jain[1] Paul Borrill[2] Chihjen Chang[3]* David Zage[3]*

[1]Independent Consultant — singhai.anjali55@gmail.com
[2]DÆDÆLUS — paul@daedaelus.com
[3]Intel Corporation — chihjen.chang@intel.com, david.zage@intel.com

netdev 0x1A — Rome — Moonshot Talk

* The views, analyses, and conclusions presented in this paper are those of the author(s) and do not necessarily reflect the official positions, policies, or opinions of Intel Corporation.

Why do we have networking?

- **Communication** — the (bi-directional) exchange of data.

Why do we have networking?

- **Communication** — the (bi-directional) exchange of data.
- **Information** — data that is *relevant and timely* for the task at hand.

Why do we have networking?

- **Communication** — the (bi-directional) exchange of data.
- **Information** — data that is *relevant and timely* for the task at hand.

The point

A network's role is not moving bits. It is to deliver *information*.

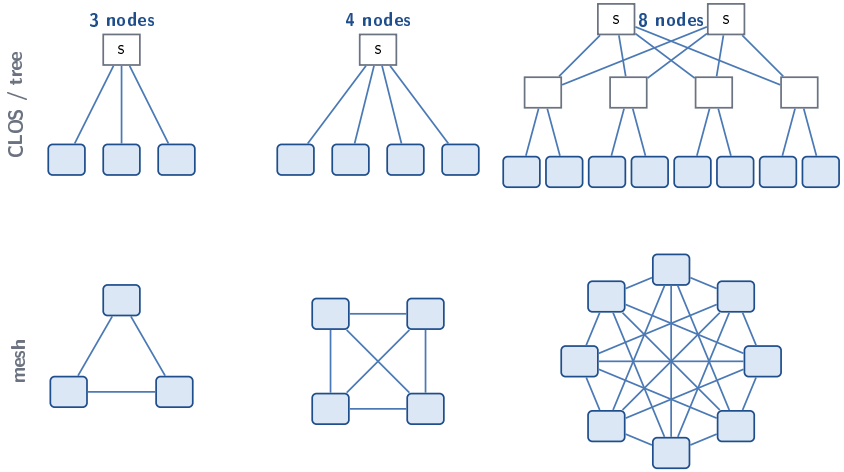
Five problems in general networking

1. **Latency** — how long until the data arrives?
2. **Jitter** — how *predictable* is that latency?
3. **Tunnel** — encapsulation & indirection hide the true path.
4. **Scheduling** — who gets the wire, and when?
5. **Time** — a shared notion of *now* to coordinate against.

The common thread

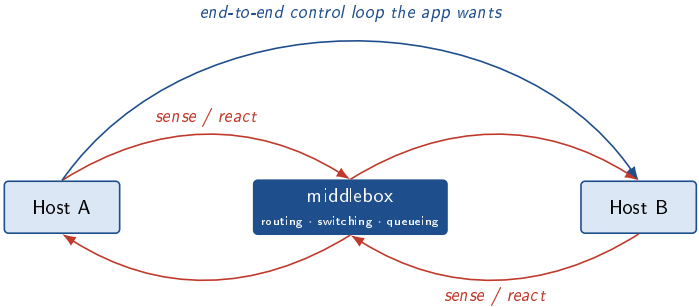
All five are really about *when*, not *whether*.

Topology matters: tree (CLOS) vs. mesh



CLOS/tree concentrates every flow onto shared spine links (a bottleneck that grows with scale); a mesh gives path diversity, so the five problems ease as nodes are added rather than worsen.

The double control-loop problem



Two nested reactive loops chase the same queue: each reacts to damage the other just caused. ***Reactive → proactive:*** *anticipate demand and agree on resource allocation before sending.*

Reactive → proactive

Middleboxes such as routing and switching (*i.e.*, the intelligence in-between) each run their own loop. Most networks are *reactive*; the fix is to become *proactive* and anticipate demand and *agree* on resource allocation before sending.

What could the next evolution look like?

congestion **control**

What could the next evolution look like?

congestion **control** $\longrightarrow$ congestion **management**

What could the next evolution look like?

congestion **control** $\longrightarrow$ congestion **management**

$\longrightarrow$ compassionate network **admission**

Agenda

1. Solutions are everywhere
2. An old idea: the end-dally
3. Compassionate promise vs imposition
4. OAE: the dally at the link layer
5. The SmartNIC's role
6. Kernel implications
7. Anticipated curiosities
8. Takeaways

Solutions are everywhere

Congestion control today is reactive

- **BBR** — infers bottleneck bandwidth & RTT from delivery.
- **ECN** — marks packets *after* a queue builds; responder reacts 1–3 RTTs later.
- **Bufferbloat / CoDel** — trades latency for capacity by managing queues.
- **PFC** — bolts lossless semantics onto a lossy substrate; mixed results.

The shared assumption

The substrate *admits speculative transmission*. The kernel's job is to detect the damage and back off in time.

The substrate invites the damage

loss → retransmit → more traffic → more loss

- Switches and routers downstream are *obliged* to absorb whatever arrives.
- Congestion control lives *above* the network, inferring trouble from indirect signals.
- We react fast because we cannot refuse.

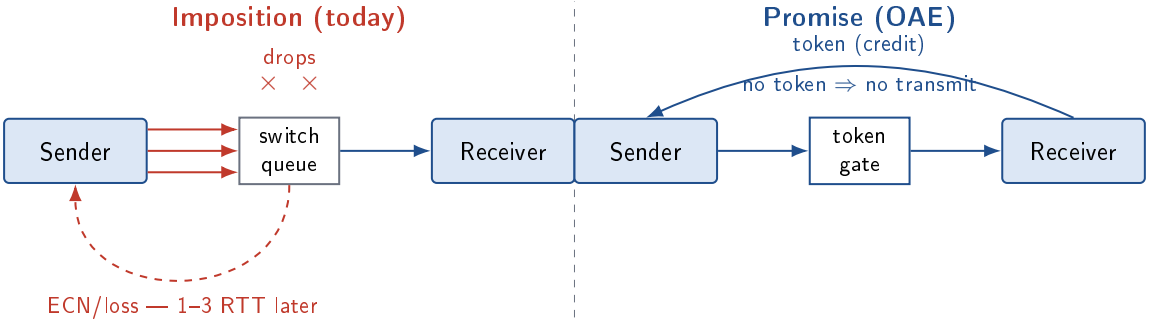
The right solution

Make speculative transmission

impossible at the source.

- Not: react faster to imposition damage.
- Instead: never admit the uninvited frame in the first place.
- Congestion control or congestion management becomes *admission management* — a property of the substrate, not a loop above it.

Two stances



Imposition: speculate, then detect damage. *Promise: gated on the peer's credit — nothing uninvited enters.*

An old idea: the end-dally

Hidden in plain sight: Metcalfe & Boggs, 1976

- Section 7.2.2 of the paper that announced Ethernet.
- The EFTP file-transfer completion handshake introduces the end-dally.
- Receiver sends END REPLY, then *dallies* — holds channel state alive — until the sender's confirming echo returns.
- The transaction is over *only* when *both* endpoints have observed the round-trip.

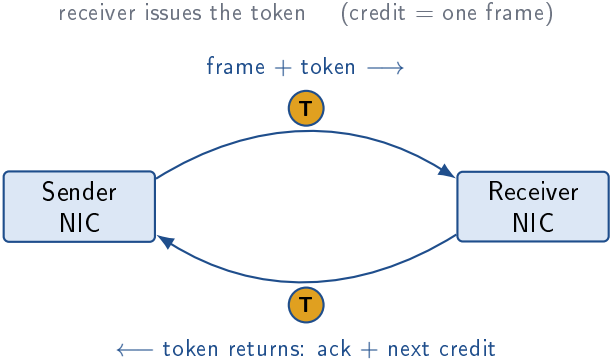
Three load-bearing properties of the dally

1. **Bilateral.** Neither endpoint can complete the transaction alone; both must observe the round-trip.
2. **A closure.** The dally interval *is* the span of unresolved mutual commitment — a hovering, not-yet-bound obligation.
3. **A performance mechanism.** Backward-flowing state (like a NAK) closes the sender/receiver gap faster than timeouts.

The dally was always bilateral — just stuck at the transport layer

- Placed at the EFTP completion boundary because that is where the protocol's *language* could express it.
- It is, in modern terms, the first deployed bilateral-commit primitive in packet networking.
- The whole move of this talk: *push the same construct down to the link layer, frame by frame.*

The dally, made physical: one frame's round-trip



*the **dally**: one frame's round-trip —*
the interval of unresolved bilateral commitment

The receiver issues a one-frame credit; the sender may transmit only while holding it; the loop is the frame's interval of unresolved bilateral commitment.

Compassionate promise vs imposition

Compassionate promise vs. imposition

Compassion

A declaration about *my own* future behaviour. I have full knowledge and control of what I promise.

Imposition

An attempt to induce behaviour in *another* agent. I have neither knowledge nor control of compliance.

- Systems and agents are **autonomous**: they cannot be forced from outside.
- An imposition is *not binding* until a matching promise to accept it binds from the other side.

Bilateral binding: a real architectural invariant

Same structure, three independent traditions:

- **Milner (CCS):** a and $\bar{a}$ synchronise into one internal τ-transition.
- **Alternating Bit Protocol:** receiver state mirrors last-acked segment.
- **OAE:** two endpoints independently prepare, then one atomic commit.

Coordination is mutual commitment, not message transfer

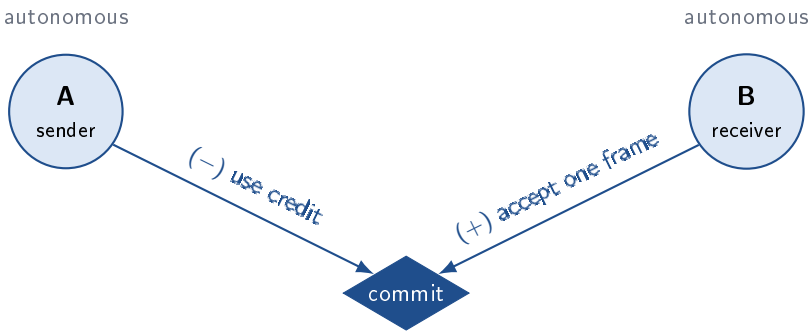
Neither side causes it alone · both must commit · it is atomic · each party witnesses it *locally*.

Global view vs. local-observer view

- **GV (global view):** reason as an omniscient observer who sees both endpoints at once. Impositions assume this.
- **LOV (local-observer view):** each endpoint has only its own state. Promises admit only this.
- **FITO** (Forward-In-Time-Only) category mistake: deploying a GV design in a LOV world.

Imposition-based congestion control is a GV pattern in a LOV world — and its pathologies are that mismatch.

Bilateral binding



neither side can cause it alone — two local commitments ordered by happens-before, not one global instant

Two autonomous agents; a (+) and a (−) promise bind into one commit — coordination is mutual commitment, not message transfer.

OAE: the dally at the link layer

The token is a physical promise

- Open Æthernet (OAE) — link-layer spec, Open Compute Project (v0.7a).
- A circulating token: a real artifact, not a metaphor.
- Issued by the *receiver*; embodies a credit of *exactly one frame*.
- A sender *must possess a token to transmit*. The token carries the data forward, then returns carrying the acknowledgment.

Permission and acknowledgment fused in a single entity.

The token round-trip *is* the dally

- Outbound token → sender spends it → data in flight → token returns.
- That interval is the dally: unresolved bilateral commitment for *one frame*.
- OAE shrinks the dally from a transport-layer *completion* construct to a link-layer *admission* construct.

Every frame has its own dally. The dally is the unit of transmission.

The structural consequence

No token, no transmit. Gated in hardware at the sending NIC.

Speculation can't enter. Saturated receiver withholds tokens; offered load = drain rate, by the physics of the link.

Bounded downstream. Queue occupancy is bounded by receiver-issued credit, not by speculative offered load.

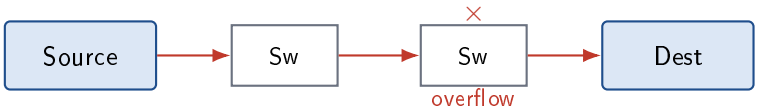
CC = admission control. The loop shrinks to *one link RTT* and acts *before* injection, not 1–3 RTTs after queue buildup.

What the token adds over prior credit

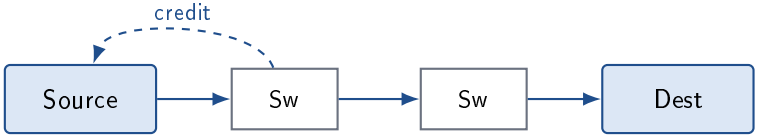
- The credit primitive is old (InfiniBand, Fibre Channel, ATM, PFC). We claim *no novelty* there.
- New is the bilateral discipline: the token is permission *and* acknowledgment, and its return is a commit both sides witness.
- Prior schemes handle the permission half without the matching ack half; a PFC pause never binds the receiver to consume.

Source-gated, bounded everywhere

Imposition: speculation fills queues downstream



Promise: gated at the source, bounded at every hop



the uninvited frame is never injected

A frame never injected at the source cannot flood a downstream queue; occupancy is bounded by receiver-issued credit.

The SmartNIC's role

From traffic policeman to first agent

- Today the SmartNIC enforces *after-the-fact* controls: based on feedback at different levels — rate limiting, shaping, ECN reaction, pacing, retransmission
- In a Promise Network it is the first promise-enforcement agent — a frame is not emitted unless the downstream node has issued a token.
- On the **SmartNIC** $\leftrightarrow$**ToR** link the ToR issues admission credits; congestion shows up not as queue growth but as the *absence of returned tokens*.

Congestion absorbed in *time*, not *memory*

- A traditional fabric absorbs congestion in memory — standing queues in switch buffers.
- A Promise Network absorbs it in time — deferred transmission opportunities at the source.
- A downstream bottleneck simply slows token circulation, and that scarcity propagates back toward the origin one hop at a time.

Credit is sized to the bandwidth-delay product — the same arithmetic as any credit scheme, the opposite stance on the fabric.

Kernel implications

Evolution vs Revolution

While it might seem like everything needs to change, this is actually an evolution

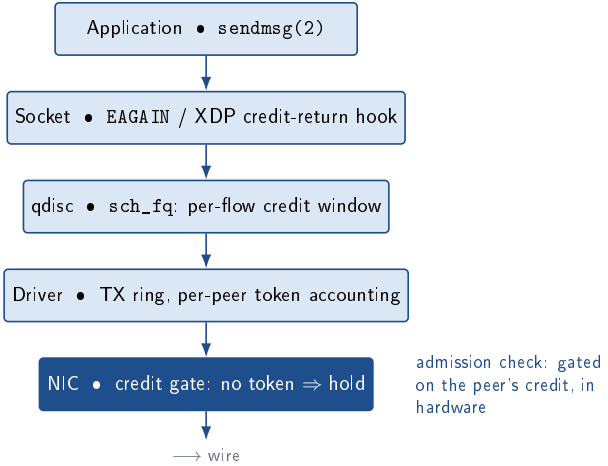
Three modest kernel surfaces

1. **Driver token accounting.** TX path consults per-peer credit before fetching from the qdisc. A cousin of BQL, but the constraint is the peer's promise.
2. **Qdisc cooperation.** `sch_fq` per-flow pacing $\rightarrow$ per-flow *credit windows*; a peer with no credit has no business at head-of-queue.
3. **Socket visibility.** `EAGAIN`/block today; optionally expose per-peer credit; an XDP hook on credit-return events.

What this is *not*

- Not a new transport. It does *not* replace TCP.
- TCP-over-OAE keeps end-to-end semantics — but runs over a substrate without the failure modes its loop was designed for.
- BBR's bottleneck becomes *observable* (token-return rate), not inferred. Transport code *simplifies*.

Where the credit gate lives



Checked lowest, at the driver/NIC boundary; surfaced upward as qdisc credit windows and socket back-pressure.

Anticipated curiosities

"You reinvented . . . "

Curiosity	**Why it's different**
Credit-based flow control	bilateral: permission *and* ack in one artifact
Receiver-driven CC (Homa, NDP)	per-link, per-frame, in hardware — not e2e grant
Token Ring	no shared medium, no arbitration; point-to-point
Bandwidth provisioning	dynamic, tracks the receiver's drain rate now

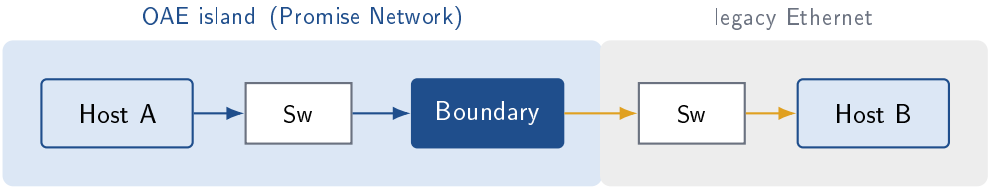
Losslessness, deadlock, liveness — keep them separate

- **Losslessness** — from the token primitive itself.
- **Deadlock freedom** — acyclic per-node spanning-tree routing + a reversible token that rolls back on a stalled dally.
- **Liveness** — RLFD detects a peer that has *stopped*, sub-millisecond. (Distinct from deadlock among *live* peers.)

Latency, errors, deployment, trust

- **"The dally costs latency."** Pay one RTT up front vs. many RTTs recovering from a wrong guess.
- **Packet errors.** A corrupted frame's credit is *never retired*; the slot is known from token accounting, not the damaged frame.
- **Heterogeneity / deployment.** Per-link; OAE islands coexist with legacy Ethernet and degrade gracefully at the boundary.
- **Security.** Token is link-local; the receiver is the accounting authority; crypto binding closes replay/forgery.

Incremental deployment



boundary node: terminates the token discipline; speculative on the legacy side — if a queue must form, it forms here

OAE is negotiated per link; contiguous OAE links form an island that degrades gracefully to legacy Ethernet at the boundary.

Takeaways

The right loop is the one you never write

- Metcalfe placed the dally at the transport-completion boundary in 1976, the only boundary his protocol could reach.
- OAE places it at *every link, every frame*.
- Promise Theory names why: the dally was always the bilateral-commit primitive; today's congestion-control corpus is its missing-bilateral form.

A Compassionately Promised Network
is a substrate in which the imposition stance is physically unrepresentable.

The ask

- We are *not* asking the kernel to commit to a wire format (v0.7a).
- We *are* asking: what should a kernel surface expose if a token-credited link layer arrives?

Thank you.

The end-dally was the first sketch of how to not write the loop.