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, Palo Alto Hills, CA, USA — singhai.anjali55@gmail.com
[2]DÆDÆLUS, Asilomar, CA, USA — paul@daedaelus.com
[3]Intel, Santa Clara, CA, USA — {chihjen.chang, david.zage}@intel.com

Abstract

Modern congestion control treats the network as a substrate onto which packets are *imposed*, and from which loss, latency, and Explicit Congestion Notification (ECN) marks must be reactively interpreted. We argue for an alternative approach that was visible in the 1976 Ethernet paper but never pushed from the transport layer down to the link layer. Metcalfe and Boggs's *end-dally* in the EFTP protocol is the bilateral closure of a transmission. Open Æthernet (OAE) generalises it into a link-layer admission primitive in which every frame is gated on a peer-issued token, so a sender without one *cannot transmit*. The consequence is structural rather than statistical: uninvited frames never enter the fabric. Congestion control then becomes an admission discipline within the network, rather than a feedback loop above it. We frame this as a Promise Theory problem and sketch the Linux kernel surface it would require. This is a moonshot proposal, presented for community critique.

Keywords

Promise Theory, bilateral protocols, congestion control, link-layer admission, end-dally, Open Æthernet, token-based flow control, Linux networking subsystem, kernel/userspace boundary, FITO category mistake.

Introduction

The Linux congestion-control corpus is large, sophisticated, and reactive. Bottleneck Bandwidth and Round-trip (BBR) models the bottleneck bandwidth and round-trip time from observed delivery [15]. Explicit Congestion Notification (ECN) marks packets after the fact at congested nodes, in the hope that a transport-layer responder reacts one to three round-trip times later [32]. Bufferbloat mitigation trades latency for capacity by tuning queue disciplines [19]. Priority-based Flow Control (PFC) attempts to push lossless semantics back onto a fundamentally lossy substrate, with mixed deployment results [22].

What this body of work shares, and is uniquely good at given the constraint, is the ability to *react fast enough* to imposition damage. The substrate admits speculative transmission. Switches and routers downstream are obliged to absorb whatever arrives. The kernel networking stack's role is to detect that the substrate is suffering and to back off in time to prevent collapse. This is a hard problem, but is it the right one? We argue that the right problem to solve is making speculative transmission *impossible at the source*.

This is not a new idea, but one hidden in plain sight since the very founding of the field. In the 1976 Metcalfe–Boggs paper that announced Ethernet, the authors introduce a construct called the *end-dally*, a bilateral-closure mechanism in their Ethernet File Transfer Protocol (EFTP) completion handshake [26]. The receiver, on receiving the final segment, sends an END REPLY and then *dallies* (*i.e.*, holds the channel state alive) for long enough to ensure the sender's own confirming END REPLY echoes back. Only when both endpoints have observed the round-trip is the transaction over. Metcalfe described the construction in his PARC video briefing [27] and reconfirmed its operational intent in recent personal correspondence [28]:

> *"As I recall, the end-dally was intended to speed things up. Like the NAK does in general."*

The dally was always a bilateral primitive by intent and by construction. It was placed at the transport-protocol completion boundary because that is where the language of EFTP could express it. Open Æthernet (OAE), the link-layer specification under active development in the Open Compute Project [31], pushes the construction down to the link layer such that every transmission is gated on possession of a circulating token and the token's round-trip is a dally. A sender without a token cannot transmit. The only data that enters the fabric is data the receiver has *already promised* to accept.

Stated in the vocabulary of Burgess's Promise Theory [9, 12, 13], a Promise Network is a network in which every link enforces bilateral promise binding before transmission and no link admits the imposition stance. The end-dally is the gating primitive that makes such a network enforceable. The consequence is that congestion control becomes admission control and a property of the substrate, not a loop above it.

This paper develops this claim for critique in one arc: it recovers the fragment of Promise Theory the argument needs, reconstructs the end-dally as the first deployed bilateral-commit primitive, shows how Open Æthernet pushes that primitive down to the link layer as a per-frame admission gate, looks at deployment from the vantage of a SmartNIC,

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

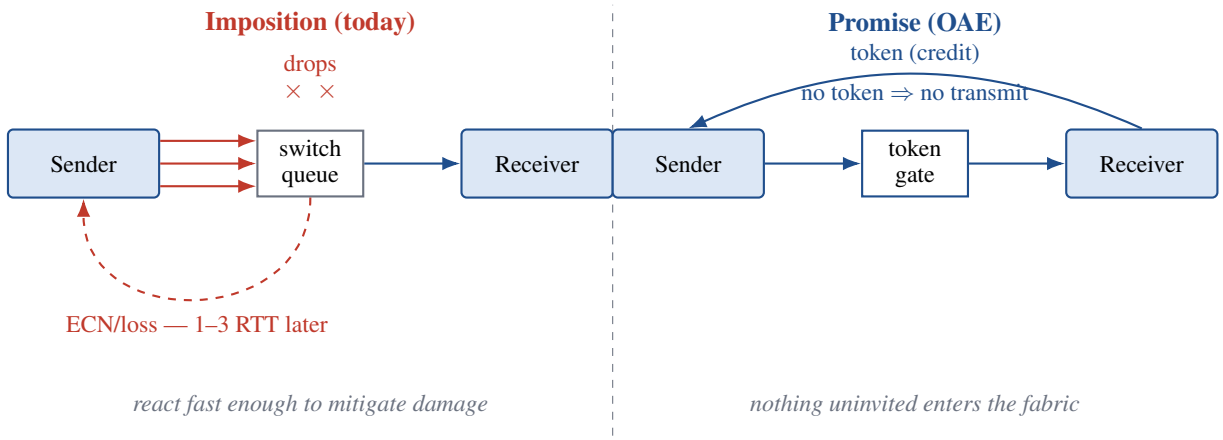

Figure 1: Understanding imposition versus promise. Under imposition networks (left), the sender transmits speculatively, the fabric absorbs what arrives and drops on overflow, and loss/ECN feedback returns one to three round-trips later. Conversely, for promise networks (right), every frame is gated on a receiver-issued token, so nothing uninvited enters the fabric.

sketches the *kernel* surface a token-credited NIC would require, and answers potential objections this community will raise.

Promise Theory: Promise, Imposition, Bilateral Commit

Promise Theory is a substantial body of work developed by Mark Burgess and Jan Bergstra, with deep roots in the production-scale CFEngine system [10, 9]. The intellectual debt of this paper is to that body of work; our contribution is only the framing of the dally as a Promise-Theoretic primitive and the link-layer consequence we draw from it. Substantive accounts of Promise Theory live in [9, 12, 11, 13, 14] and this section recovers only the fragment the paper needs.

Autonomous Agents

The foundational concept is the *autonomous agent*. An agent is causally independent (its behaviour originates from within and cannot be *caused* by external agents) and decision-independent (it makes its own choices about which promises to make and which to keep). As noted in [9]: "Autonomous meant two things: causally independent, and independent in all decisions. Agents couldn't be forced by other agents".

This is not a stylistic preference. It is a recognition that any model in which one agent's *command* is treated as another agent's *guaranteed behaviour* has a hidden assumption: that compliance is enforced from outside the model. In real distributed systems, and especially in kernel networking, where the peer endpoint is by construction not under the local kernel's control, this assumption fails silently. The autonomy principle is the formal expression of the rule that the only behaviour the model can predict is the behaviour the agent has actually committed to.

Promises and Impositions

Promise Theory distinguishes two forms of declared intention. A *promise* is an agent's declaration about its own future behaviour; the promiser has complete knowledge and control over what is promised. An *imposition* is an attempt to induce behaviour in another agent; the imposer has neither knowledge nor control over compliance. An imposition is not binding until the target agent makes a corresponding promise to accept it.

In notation, a promise from agent A to agent B with body b is written $A \xrightarrow{b} B$, and an imposition is written $A \xrightarrow{b}\!\blacksquare\, B$. Promises and impositions carry a polarity: a promise to give behaviour b has body $+b$, a promise to accept has body $-b$; correspondingly for impositions. *An imposition is not satisfied until a matching* $(-)$ *promise binds to it from the target side.*

Bilateral Binding

The bilateral-binding structure is the crucial mechanism of Promise Theory and the technical core of this paper. The same structure recurs in:

- Milner's Calculus of Communicating Systems, where complementary actions a and $\bar{a}$ synchronise into an internal τ-transition only when both are simultaneously enabled [29]
- The Alternating Bit Protocol, where the receiver's bit-state reflects the sender's last-acknowledged segment [25, 2]
- The bilateral commit of OAE, where two endpoints must independently prepare a contribution before a single atomic commit event occurs [31].

The three-way structural convergence of these works suggests a real architectural invariant that *coordination is mutual commitment, not message transfer*. In each case, four properties hold:

1. Neither party can unilaterally cause the interaction.
2. Both parties must autonomously commit.
3. The interaction is atomic: there is no partial binding.

4. The result is a mutual commitment that each party witnesses locally and that becomes common knowledge only through the round-trip, not a message in flight observed by one side alone.

The "single atomic commit" is a modeling abstraction. In Milner's interleaving semantics, it is one internal τ-transition. Under a strictly local view, where each endpoint sees only its own observations, it is realised as two local commitments ordered by happens-before, never as a single globally observed instant. Bilateral binding is precisely what closes that gap over the dally.

Global vs. Local Assessment

A standing temptation in network design is to reason about a system from the perspective of an omniscient observer who sees both endpoints' states simultaneously. We call this the global view (GV) and contrast it with the local-observer view (LOV), in which each endpoint has only its own local state and observations. Impositions assume GV, where a command can be issued because the commander implicitly sees the commanded agent's state. Promises admit only LOV, since each agent declares only what it itself will do. Any apparent global property must then emerge from bilateral reconciliation between specific pairs of endpoints.

The FITO (Forward-In-Time-Only) category mistake [8] is the failure mode that arises when a system designed under GV assumptions is deployed in a world that only admits LOV. The vocabulary will recur in the *Open Æthernet* section: imposition-based congestion control is a GV design pattern in a LOV world, and its pathologies are precisely the consequences of that mismatch.

The End-Dally: Bilateral Closure in Ethernet

Section 7.2.2 of the Metcalfe–Boggs [26] describes the completion handshake of the Ethernet File Transfer Protocol (EFTP). When a sender has dispatched the final segment of a file, the protocol issues an END packet bearing the next-in-sequence number. The receiver responds with an END REPLY and then *dallies*: it holds the channel state alive for a long-enough interval that the sender, on receiving the END REPLY, can echo a confirming END REPLY back, and the dallying receiver can observe that echo. As described by Metcalfe [27]:

> *"... the protocol also calls for this machine having received the end and sent the end reply to dally, that is, it just sits there. Instead of going away having completed the file transfer, it stays on the off chance that another, that the end reply will be lost... So when a dallying receiver receives an end reply, it can then safely go away, the transaction having been completed."*

What the Dally Actually Is

Three properties of the end-dally are load-bearing.

1. The dally is bilateral. Neither endpoint can complete the transaction unilaterally. The sender does not know the receiver has the file until it observes the END REPLY. The receiver does not know the sender knows it has the file until it observes the echoed END REPLY. Both endpoints must independently observe the round-trip before either can declare the transfer complete. This is precisely Promise Theory's bilateral-binding structure.

2. The dally is a closure. The interval of the dally is the interval of *unresolved bilateral commitment*, the time during which one endpoint has progressed but does not yet know whether the other knows it has progressed. The end-dally ends when both endpoints have observed the round-trip; this is the moment at which bilateral commitment crystallizes. The interval has a name precisely because it is not nothing: it is the duration of a hovering, not-yet-bound, mutual obligation [4].

3. The dally is a performance mechanism. Metcalfe's spontaneous comparison of the end-dally to the NAK [28] is the diagnostic clue. NAK-based protocols are faster than ACK-timeout protocols precisely because backward-flowing information (*i.e.*, the receiver actively reporting state to the sender) closes the epistemic gap between sender and receiver faster than hopeful silence ever could. The dally is the bilateral correlate: it forces the sender to confirm receipt of the receiver's acknowledgment, closing the loop in both directions.

The Dally as a Promise-Theoretic Primitive

Translated into Promise Theory vocabulary, the receiver's END REPLY is a promise $(+)$ to acknowledge while the sender's echoed END REPLY is a promise $(-)$ to use that acknowledgment. The dally is the interval during which the bilateral binding has not yet completed, it is the time the receiver-side promise has been issued but not yet matched. The end of the dally is the moment of bilateral commit: the moment, in Promise Theory's structural sense, at which both polarities have bound and each endpoint has locally observed the round-trip.

Previously, this lived at the transport layer because EFTP was a transport protocol and the language available was the language of file-transfer completion. The argument of the next section is that the same construction pushed down to the link layer is the enabling primitive of a congestion-free fabric.

Open Æthernet: Dally at the Link Layer

Open Æthernet (OAE) is a link-layer specification in active development under the Open Compute Project [31]. OAE's reliability mechanism is a circulating token, a protocol artifact rather than a metaphor, that carries permission and acknowledgment in a single entity. Each token is issued by the receiver, embodies credit for exactly one frame, and circulates continuously between the two endpoints of a link. A sender must possess a token to transmit. The token then returns to its issuing endpoint with acknowledgment of receipt embedded in its return path.

The Token as Physical Promise

The token is a realisation of the receiver's promise $(+)$ to accept one frame and the sender's act of consuming the token by transmitting is the materialisation of its matching promise $(-)$ to use that credit. The token's return is the bilateral commit observable to both endpoints. Every transmission on an OAE link is therefore a complete bilateral binding by construction. There is no act of injection that is *not* a binding.

The round-trip of the token is, in Metcalfe's vocabulary, a dally. The interval during which the receiver-issued token is outbound to the sender, the sender is consuming it, the data is in flight, and the token is returning is the interval of unresolved bilateral commitment for that single frame. OAE reduces 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

The consequence for congestion control is not statistical. It is structural.

A sender without a token cannot transmit. The act of transmission is gated on possession of a token issued by the peer. If no token is in the sender's hand, no frame leaves the sending NIC. This is a hardware-level constraint, not a software policy.

Speculative or excess frames cannot enter the fabric. If the receiver is saturated, it withholds tokens. The sender's offered load is rate-limited to exactly the receiver's drain rate, by the physics of the link. There is no buffer to fill at the sender beyond the in-flight tokens; there is no transmission opportunity to overrun.

Downstream hops never see speculative traffic. The most consequential property: a frame that was never injected at the source cannot appear at hop two, hop three, or hop n. Queue occupancy at every downstream node is bounded by the cumulative receiver-issued credit of the path, not by the cumulative speculative offered load of the senders.

Congestion control is admission control. What BBR, ECN, and qdisc-level pacing *infer* from indirect signals (*e.g.*, loss, RTT inflation, marked packets) one to three round-trips after the damage has begun, the dally *enforces* at the source one frame at a time, before damage can begin. The feedback loop is reduced to its minimum, a one link RTT, which acts before injection rather than after queue buildup. Within that one-RTT loop the receiver bounds the sender's offered load directly, via the issuance and withholding of tokens.

What the Token Does That Prior Credit Schemes Did Not

The reader will note that the credit-based-flow-control literature is rich. InfiniBand link-level credit return [23], Fibre Channel buffer-to-buffer credits [1], the credit-based ATM work of Kung and others [24], and the IEEE 802.1Qbb Priority Flow Control [22] all use a credit token of some kind to gate transmission. We do not claim novelty for the credit-token primitive itself. What OAE provides that the prior corpus did not is the *bilateral* discipline: the token is not merely a permission, it is also an acknowledgment, and its return path is the bilateral commit observable to both endpoints. The asymmetric credit schemes of InfiniBand and Fibre Channel handle the permission half of the binding without the matching acknowledgment half; PFC pauses the sender but does not bind the receiver to a corresponding promise of consumption. We address the distinction in detail in *Anticipated Objections*.

A more recent line of datacenter work brings receiver-issued credit up to the transport and network layers: ExpressPass schedules credit packets end-to-end (E2E) to bound queueing [16], while Homa [30], NDP [20], and pHost [18] are receiver-driven transports in which the receiver grants transmission before the sender injects. These share the dally's receiver-grants-first stance; *Anticipated Objections* addresses what a link-layer bilateral token adds over them.

The Reliable Link Failure Detector

The OAE specification [31] defines the Reliable Link Failure Detector (RLFD) as the NIC-level mechanism that detects link failures bilaterally and sub-millisecond. It is built from four primitives: Perfect Information Feedback (PIF), a committed transfer, Triangle Failover, and Atomic Token Transfer. In Promise Theory terms, RLFD is the local assessment mechanism that maintains the link-level superagent's promise: it lets each endpoint locally evaluate whether the peer is keeping its end of the bilateral binding. Triangle Failover is the mechanism by which the superagent maintains its own promise to higher layers even when one of its constituent endpoints becomes unresponsive. The point for the present paper is that RLFD closes the only remaining failure mode that token-gated admission alone does not close: the case where the bilateral peer disappears. Without RLFD, a sender holding a token for a peer that has silently gone away could wait indefinitely; with it, the local assessment converges on link failure within milliseconds.

The SmartNIC Perspective

The discussion thus far has described Promise Networks as an idea between adjacent endpoints. In a datacenter, the device that makes it concrete is the SmartNIC. Congestion originates not inside transport protocols but at the interface between software demand and network admission. Applications, virtual machines, containers, and storage stacks generate traffic based on local incentives and the NIC is the first point at which that demand becomes frames. Today the SmartNIC enforces after-the-fact controls including rate limiting, shaping, ECN reaction, congestion-window manipulation, pacing, and retransmission. In a Promise Network it instead becomes the promise-enforcement agent such that a frame is not emitted unless the downstream node has already issued a token authorizing it. The token is thus an architectural contract binding the SmartNIC's computational domain to the network's admission domain.

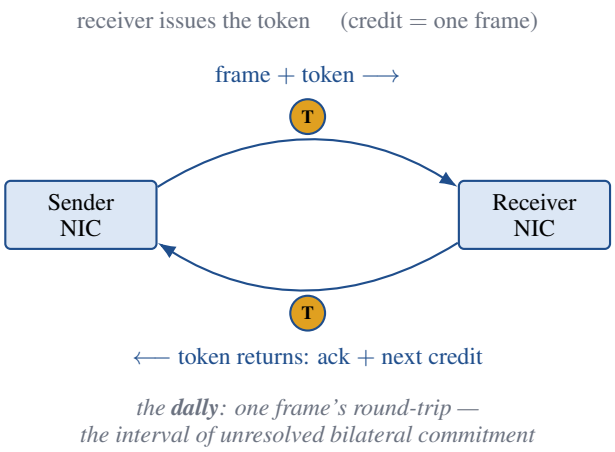

Figure 2: The token round-trip *is* the dally. The receiver issues a token (a credit for exactly one frame); the sender may transmit only while holding it; the token carries the frame forward and returns with the acknowledgment and next credit. The full loop is the interval of unresolved bilateral commitment for that frame.

Direct SmartNIC-to-ToR Promise Links

The most immediately valuable deployment is the bilateral link between a SmartNIC and its Top-of-Rack (ToR) switch. A server may carry hundreds or thousands of active flows that the ToR sees only once their frames have entered the fabric (*i.e.*, only after resources are already consumed). In a Promise Network, the ToR is the issuer of admission credits where each SmartNIC–ToR link carries a pool of circulating tokens, one per frame of fabric capacity, issued at the rate the ToR can absorb and forward. Congestion no longer manifests as queue growth inside the network but as the absence of returned tokens. Capacity is communicated directly rather than inferred (*e.g.*, no retransmissions to repair congestion-induced loss) and the congestion signal becomes a first-class protocol object rather than a statistical inference.

Congestion Absorbed by Time, Not Memory

Token pools are sized to the bandwidth-delay product (BDP) of each segment with the sizing arithmetic being the same as any credit scheme. Unlike a TCP congestion window, these credits are provisioned from known physical properties of the path rather than expanded until loss and then collapsed reactively. The effect is analogous to replacing an uncontrolled freeway on-ramp with a metered admission gate where traffic volume is unchanged, but entry is paced to capacity. The distinctive property is where the congestion goes. A traditional network absorbs congestion in memory while a Promise Network absorbs it in time as 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, so admission limits compose hop-by-hop without any node holding a GV of the path. Congestion is not eliminated in that demand can still exceed supply, but it does not accumulate inside the fabric.

Incremental Deployment

OAE is negotiated per link at bring-up, in the same way link partners already negotiate speed, duplex, PFC, or a TSN feature set. A partner that does not speak OAE causes the link to fall back to conventional Ethernet, so a token-credited link and a legacy link coexist on the same path without special arrangement.

The Promise Network properties hold across each contiguous span of OAE-capable links (*i.e.*, an island) and degrade gracefully at the boundary to a legacy segment, where end-to-end transports revert to current protocols. A boundary node terminates the token discipline on its OAE side and behaves as an ordinary speculative sender on its legacy side. If a queue must form anywhere, it forms at that boundary, which is exactly where an operator would choose to place it.

Adoption is therefore incremental. An operator deploys Promise Networks where it pays most (*i.e.*, from the ToR to the SmartNIC) and expands the island outward with no forklift upgrade. We will see how this could map to the kernel in the next section.

Kernel Implications

A token-credited link layer does not retrofit transparently into the existing Linux networking stack. `sendmsg(2)` returns when the kernel has accepted the buffer into an egress queue, regardless of whether any peer has bound a corresponding promise to receive it. The qdisc layer schedules onto the wire on its own clock. The driver enqueues to TX rings under the assumption that the medium is always available. None of these surfaces expose the receiver-issued-credit constraint that the dally requires.

This section sketches three modest kernel surfaces that would close the gap. We do not claim a finished design; the moonshot ask is for community review of where in the stack the gating belongs.

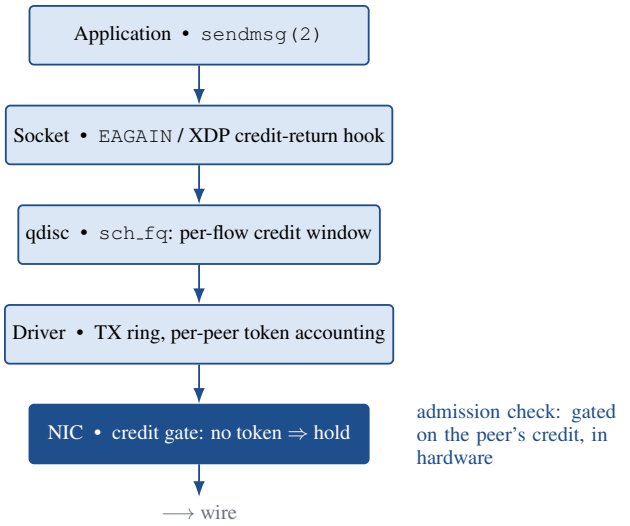

Figure 3: Where the admission discipline lives in the stack. The receiver-issued-credit constraint is checked lowest, at the driver/NIC boundary (no token ⇒ hold), and surfaces upward as per-flow credit windows in the qdisc and back-pressure at the socket.

Driver-Level Token Accounting

The most localised change is in the NIC driver. A token-credited NIC exposes, per active peer, the number of currently held tokens and the rate at which tokens are returning. The driver's TX path consults this credit before fetching a buffer from the qdisc; if no token is available for the target peer, the buffer is not fetched. This is a small extension to existing per-queue back-pressure (TX-ring-full flow control, BQL byte queue limits) but with a critical difference: the constraint is not "my own queue is full" but "the peer has not promised to accept".

Qdisc Cooperation

For per-flow fairness across multiple peers sharing a NIC, the qdisc must learn that the wire is not the only contention point. `sch_fq`'s pacing primitives [17] already maintain per-flow next-transmit times; the natural extension is per-flow credit windows that throttle dequeue to the receiver-issued rate. `sch_fq_codel` would similarly benefit from a credit-aware admission test: a flow whose peer has no credit available has no business holding head-of-queue, regardless of Controlled Delay's drop policy.

Socket-Layer Visibility

The application-visible question is what `sendmsg(2)` should return when the peer has no credit. The minimal answer is the existing back-pressure semantics: block on `SOCK_STREAM`, return `EAGAIN` on `SOCK_NONBLOCK`. The more interesting answer, which we do not commit to but flag for discussion, is a new socket-option family that exposes the per-peer credit count as a queryable property, so applications can adapt offered-load policy at the source rather than blindly retrying. A natural shape is an XDP-program hook [21] that fires on credit-return events, letting userspace observe the receiver's pacing in real time.

What This Is Not

This is not a new transport protocol. It does not replace TCP. TCP-over-OAE retains its end-to-end semantics; what changes is that TCP's congestion-control loop now operates over a substrate that does not present the failure modes its loop was designed for. BBR's model of the bottleneck bandwidth becomes simpler because the bottleneck bandwidth is now *observable as the receiver's token-return rate* rather than inferred from delivery dynamics. ECN marking becomes trustworthy because there is no longer a loss signal to compete with. We expect transport-layer code to simplify, not to be replaced.

Nor is the bilateral-commit stance confined to the wire: the same discipline one boundary up, at the kernel/userspace interface, is developed in a companion Netdev 0x1A paper on bilateral syscalls [3]. Whether the boundary is a link or a syscall, an operation is not complete until both sides have witnessed its completion.

Anticipated Objections

"You Reinvented Credit-Based Flow Control"

The prior corpus on credit-based flow control, including InfiniBand link-level credit return, Fibre Channel buffer-to-buffer credits, and PFC, is substantial, and we acknowledge it explicitly [23, 1, 24, 22]. The OAE token differs in three ways relevant to the present argument.

First, the token is *bilateral*: its issuance is a $(+)$-promise from the receiver, its consumption is a $(-)$-promise from the sender, and its return is the bilateral commit event observable to both. InfiniBand and Fibre Channel credits are receiver-asserted but not symmetrically acknowledged; the receiver issues a credit, the sender consumes it, but the link does not maintain a single artifact whose round-trip is the unit of mutual commitment.

Second, the token carries acknowledgment as well as permission. There is no separate ACK path; the same artifact that authorises the next frame also signals receipt of the previous one. The protocol cannot get into a state where data has arrived but acknowledgment has not, because the artifact that carries the acknowledgment is the same one that enabled the data. The regress that one-way acknowledgments cannot escape — each ACK needing its own ACK, without end — is analysed in a companion IEEE 802 Nendica contribution [5].

Third, OAE pairs the token with RLFD-level liveness assessment (the *Open Æthernet* section), so a peer that disappears with tokens outstanding is detected at the link level in sub-millisecond time. PFC pause frames have no equivalent liveness mechanism. Deadlocked PFC pause storms have caused production outages [22]. Three concerns should not be conflated: losslessness, deadlock freedom, and liveness. Losslessness is supplied by the token primitive itself. RLFD addresses liveness while deadlock among live peers is a separate hazard every lossless credit scheme must design out. OAE keeps the buffer-dependency graph acyclic by construction, routing over per-node spanning trees so that

no cyclic wait can form, and its reversible token model rolls back any transfer whose return does not close within the dally, so a stalled exchange resolves rather than persists. Deadlock avoidance is thus a property of the routing and reversibility discipline, distinct from both the losslessness the token supplies and the liveness RLFD supplies. The underlying trilemma — losslessness, deadlock freedom, arbitrary topology: pick two — is developed as a first-principles analysis of PFC-based fabrics in a companion IEEE 802 Nendica contribution [6].

The relation of OAE tokens to prior credit schemes, then, is one of generalisation and discipline. The credit primitive is shared, the bilateral discipline is what is new.

"You Reinvented Receiver-Driven Congestion Control"

The closest modern relatives of the dally are the receiver-driven datacenter transports: ExpressPass [16], Homa [30], NDP [20], and pHost [18]. Each has the receiver grant transmission before the sender injects, and each is, in our vocabulary, an attempt to recover the dally's stance from above. The distinction is where the grant lives. In these systems, the grant is a transport- or network-layer message that traverses the whole path. It costs an end-to-end round-trip and gates the source on the *destination's* willingness rather than each link's. The substrate also remains fundamentally lossy, so when a grant estimate collides with in-network contention, the transport falls back to loss and timeout. When many senders time out together, synchronized retransmissions feed the very congestion that triggered the estimate, the storm dynamics quantified in [7]. The OAE token is per-link and gated in hardware on the immediate link partner's credit, with permission and acknowledgment fused in a single artifact whose round-trip is the bilateral commit. Admission composes hop-by-hop with no endpoint holding an end-to-end reservation, and speculative injection is unrepresentable at *every* link, not merely discouraged end-to-end. Receiver-driven transports attempt to recover the dally's discipline from above; OAE enforces it from below.

"You Reinvented Token Rings"

A token ring is a *shared-medium arbitration* scheme in which one token circulates among N stations on a common medium and confers the right to transmit *onto the medium*. It answers "is the bus free, and is it my turn?" There is a contention domain, a MAC arbitration discipline, and a ring of stations sharing one channel.

The OAE token is the opposite kind of object. It is *per-link and bilateral*: a private artifact circulating between exactly the two endpoints of one point-to-point link. It is issued by the *receiver*, embodies a credit of exactly one frame of buffer the receiver has promised to accept, and carries permission and acknowledgment in a single entity. It answers a different question: "may I send one frame to *you*, and did *you* receive the last one?" There is no shared medium, no contention domain, and no arbitration among stations.

The "ring" visible in the Tx/Rx path is real, but it is the round-trip of one credit on one link (*i.e.*, a dally). It is ring-shaped at the level of one link's round-trip, and *not* a token ring in the 802.5 sense. The token is a bilateral-commit primitive, not a medium-arbitration baton.

"Isn't This Just Bandwidth Provisioning?"

To reach line rate, the outstanding-credit window is sized to the link's *bandwidth-delay product*, so that there are always enough credits in flight to keep the wire busy while the earliest credits are still returning. This is standard credit-based-flow-control sizing, the same arithmetic as Fibre Channel buffer-to-buffer credits or InfiniBand link-level credit return.

The difference from a provisioned or reserved-bandwidth network is two-fold. First, the credit is dynamic and bilateral. It tracks the receiver's *actual instantaneous drain rate*, frame by frame, rather than a statically reserved rate negotiated up front. There is no admission-control reservation to negotiate, no over-provisioning headroom to waste, and no under-provisioning that throttles a burst the receiver could have absorbed. The receiver's true capacity, right now, *is* the rate.

Second, a provisioned network reserves capacity in the fabric and then admits speculative traffic up to that reservation, policing after the fact. OAE reserves *nothing* in the fabric; it gates at the source, so the fabric never carries uninvited traffic in the first place. Provisioning is the GV stance: reserve globally, then hope the reservation holds. OAE is the LOV stance: the receiver paces the sender directly. The control loop is exactly one link's round-trip and it acts before injection rather than after the fact. The result is the same bandwidth-delay-product arithmetic for window sizing, but the opposite stance on the fabric. A provisioned network still admits speculative transmission, up to the reservation, and cleans up after; OAE makes speculative transmission structurally impossible.

"The Dally Costs Latency"

A reader steeped in high-bandwidth-delay-product reasoning will object that requiring possession of a receiver-issued token before transmission introduces a round-trip of latency that loss-tolerant speculative transmission avoids. The objection is real but its conclusion is wrong.

The speculative-transmission stance pays the round-trip of latency for every detected loss in the form of a retransmission, a congestion-window collapse, and a recovery period. It pays this cost on the inferred (post-hoc) signal rather than on the actual (pre-imposition) constraint. The dally pays the round-trip once, at the start, to know whether the receiver is ready; the speculative stance pays it many times, after the fact, to recover from a guess that was wrong.

Empirically, this manifests as the well-known difference between loss-driven congestion control and credit-driven flow control: at saturation, the loss-driven path oscillates around the operating point with multi-RTT recovery loops, while the credit-driven path tracks the receiver's drain rate exactly. The 2017 BBR paper [15] reads as a sophisticated attempt to recover credit-driven behaviour from loss-driven signals; the OAE stance recovers it by construction, without inference.

"What About Packet Errors?"

The answer relates to *whose credit the token is*. A token is the receiver's promise to accept *one* frame into *one* buffer

slot. When a frame is dropped at the receiver (bit error, CRC failure), the receiver does *not* retire that credit as consumed. The same bilateral-commit event that would have signalled "got it" instead signals "this slot is still owed" and the credit returns to the sender as still available for that frame. The errored frame's retransmission is therefore gated on a credit that was never actually spent. The slot is identified by the token accounting rather than by any field the corrupted frame carries. The receiver knows which of its outstanding credits is still unmatched because it issued that credit and has not observed its matching return, so attribution survives even payload corruption. If the receiver observes no arrival at all, the sender's own dally for that slot expires and reclaims the still-unspent credit.

The reason other queued frames cannot steal a token is that credit is not a single global baton. The sender holds a *window of outstanding tokens*. A retransmit reuses the credit for the slot the receiver is still holding open and it does not queue behind the pipeline waiting for a single token to come around.

"Does This Require a Homogeneous Network?"

The mechanism is per-link by design, and that is a *feature*, not a homogeneity requirement. Each hop independently enforces its own bilateral admission: hop k issues credit to hop $k-1$ at hop k's actual drain rate. Back-pressure is therefore local and transitive; it composes hop by hop without requiring a uniform fabric.

Downstream queue occupancy at every node is bounded by the receiver-issued credit, not by the speculative offered load of the senders. Every link need not be identical but only needs to honour the discipline. Heterogeneous links with different rates and different buffer depths compose cleanly, because a slow link simply issues credit more slowly and that scarcity propagates back toward the source one hop at a time. The slow hop paces the fast hop; the fast hop never floods the slow one because it was never credited to do so.

The one failure mode token-gated admission does *not* close by itself is a peer that disappears where a sender is left holding a token for a peer that has silently gone away. That is exactly what the Reliable Link Failure Detector (RLFD) handles.

The substrate holds per link, not per path: a Promise Network does not extend end-to-end across the Internet. Transports composed over it see loss-from-congestion removed as a failure mode, while loss-from-bit-error and loss-from-link-failure remain (the latter detected by RLFD), so TCP-friendly fairness stays a property of the end-to-end transport, not of the link. The dally changes what the substrate offers, not what the transport must do above it.

"What Prevents Malicious Actors from Gaming Tokens?"

Three attacks are worth naming. A sender could *forge* a token, minting credit it was never issued, whether through a compromised or a merely buggy NIC. A sender could *hoard* tokens, withholding the credits it owes its peer and starving the reverse direction. Or an attacker could *replay* a captured token to claim the same credit twice.

What bounds all three is that the token is a link-local object. The peer is not a remote host somewhere on the Internet but a peer one hop away, so the trust boundary is physical adjacency. This is the same boundary on which PFC or an 802.1X-controlled port already relies. The receiver is the accounting authority because it issues credit and counts its own outstanding tokens, so a token it never issued is simply rejected. Forgery requires an actual hardware compromise of the immediate link partner, not just a misbehaving distant sender. Hoarding is a liveness failure, and it is exactly the case RLFD already detects: a peer that stops returning tokens is flagged within milliseconds.

Replay and forgery could be prevented outright by binding each token cryptographically to its issuer, an epoch, and a sequence number, at some cost in per-frame latency that we leave to future work. Even without it, token-gated admission is already strictly stronger than the imposition stance it replaces in which any host may inject speculatively.

"Why Should Linux Care About a Link Layer That Is Not Yet Shipping?"

The honest answer is that the OAE specification is at v0.7a [31] and we are not yet asking the kernel to commit to a wire format. The kernel-side discussion we are inviting is about the *primitive*: what does a kernel surface need to expose if a token-credited link layer arrives? Our position is that this is exactly the right moment to have that conversation, before three competing approaches accrete in three subtly incompatible forms in three subsystems, and before vendor-specific extensions to existing flow-control hardware create a fait accompli that the kernel must then accommodate.

Conclusion

The Linux networking subsystem has built, over four decades, an extraordinary collection of mechanisms to react to substrate damage. BBR estimates the bottleneck. ECN signals congestion. CoDel manages buffer occupancy. Pacing smooths bursts. Each of these is sound engineering in the imposition stance. Each of them exists because the substrate does not on its own refuse to admit speculative transmission.

The argument of this paper is that the substrate should refuse. The dally was originally placed at the transport-completion boundary in 1976, and Open Æthernet now places it at every link, frame by frame. Promise Theory gives us the vocabulary to recognise that the dally was always the bilateral-commit primitive, and that the post-hoc congestion-control corpus is its missing-bilateral form. A Promise Network is not a metaphor, but a substrate in which the imposition stance is physically unrepresentable and the kernel's networking surfaces above can be simpler and smaller than imposition ever let them be.

The right congestion-control loop is the one you never have to write. The end-dally was the first sketch of how to not write it. OAE is the first link layer to take this seriously. The Linux kernel networking community is the integration point at which the architectural recognition can happen.

Concurrent standards activity. The link-layer program described here has been contributed to the IEEE 802 Nen-

dica activity as a first-principles series on network efficiency, three parts of which are cited above [7, 5, 6].

We present this paper for the community's critique.

Acknowledgments

Our primary debt is to Mark Burgess, whose Promise Theory supplies the formal vocabulary throughout [9, 12, 11, 13, 14] and the 2014 draft *A Promise Theory Perspective on Data Networks*, with Burgess, Todd Craw, and Mike Dvorkin, is the ancestor of the *Promise Theory* section. The end-dally material owes its primary source to Bob Metcalfe, who invented the construction in 1976 [26] and reconfirmed its bilateral intent in 2026 correspondence [28]. Neither has seen this paper nor bears responsibility for its framing.

The OAE specification is the collective work of the OCP Open Æthernet Working Group. The companion paper *The Compassionate Network* (Singhai-Jain & Borrill), under separate submission to this venue, supplied editorial and notational scaffolding; the two are independent in their arguments.

AI Assistance Disclosure

The authors used Claude (Anthropic) as a drafting and editing assistant for prose, citation hygiene, and structural review. All technical claims, attributions, and editorial judgements are the authors'. The argument of the paper that the end-dally is a bilateral-commit primitive whose link-layer generalisation collapses the congestion-control problem into an admission-control problem is owned by the authors.

References

[1] ANSI INCITS. 2017. Fibre Channel-fs-3: Framing and signaling 3, buffer-to-buffer credit.

[2] Bartlett, K. A.; Scantlebury, R. A.; and Wilkinson, P. T. 1969. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM* 12(5).

[3] Borrill, P., and Singhai, M. 2026. Why syscalls fail: Bilateral commit at the linux kernel/userspace boundary. Netdev 0x1A, Rome, July 2026.

[4] Borrill, P. 2026a. The bilateral efficiency of ethernet: Recalibrating metcalfe and boggs after fifty years. arXiv:2603.19406.

[5] Borrill, P. 2026b. The cost of confirming (OAE network efficiency series, part VII). IEEE 802 Nendica contribution, DCN 1-26-0024-00, mentor.ieee.org.

[6] Borrill, P. 2026c. Lossless, deadlock-free, any topology: Pick two (OAE network efficiency series, part VIII). IEEE 802 Nendica contribution, DCN 1-26-0025-00, mentor.ieee.org.

[7] Borrill, P. 2026d. Timeout-and-retry storms (OAE network efficiency series, part VI). IEEE 802 Nendica contribution, DCN 1-26-0021-01, mentor.ieee.org.

[8] Borrill, P. 2026e. What distributed computing got wrong: The category mistake that turned design choices into laws of nature. arXiv:2602.18723.

[9] Burgess, M., and Bergstra, J. A. 2019. *Promise Theory: Principles and Applications*. XtAxis Press, second edition. First edition 2014.

[10] Burgess, M. 1995. CFEngine: A site configuration engine. In *USENIX Computing Systems*, volume 8(3).

[11] Burgess, M. 2013. *In Search of Certainty: The Science of Our Information Infrastructure*. XtAxis Press.

[12] Burgess, M. 2015. *Thinking in Promises: Designing Systems for Cooperation*. O'Reilly Media.

[13] Burgess, M. 2018. Promise Theory of networking and distributed systems (long form). arXiv:1807.08549.

[14] Burgess, M. 2022. Information and causality in promise theory. arXiv:2204.00470.

[15] Cardwell, N.; Cheng, Y.; Gunn, C. S.; Yeganeh, S. H.; and Jacobson, V. 2017. BBR: Congestion-based congestion control. *Communications of the ACM* 60(2):58–66.

[16] Cho, I.; Jang, K.; and Han, D. 2017. Credit-scheduled delay-bounded congestion control for datacenters. In *ACM SIGCOMM*.

[17] Dumazet, E. 2013. FQ and pacing in the linux kernel (sch_fq). kernel.org documentation.

[18] Gao, P. X.; Narayan, A.; Kumar, G.; Agarwal, R.; Ratnasamy, S.; and Shenker, S. 2015. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *ACM CoNEXT*.

[19] Gettys, J., and Nichols, K. 2011. Bufferbloat: Dark buffers in the Internet. *Communications of the ACM* 55(1):57–65.

[20] Handley, M.; Raiciu, C.; Agache, A.; Voinescu, A.; Moore, A. W.; Antichi, G.; and Wójcik, M. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM*.

[21] Høiland-Jørgensen, T.; Brouer, J. D.; Borkmann, D.; Fastabend, J.; Herbert, T.; Ahern, D.; and Miller, D. 2018. The eXpress data path: Fast programmable packet processing in the operating system kernel. ACM CoNEXT.

[22] IEEE 802.1Qbb Working Group. 2011. IEEE 802.1qbb-2011: Priority-based flow control.

[23] InfiniBand Trade Association. 2022. InfiniBand architecture specification, volume 1, section 7.9 (link-level flow control).

[24] Kung, H. T.; Blackwell, T.; and Chapman, A. 1995. Credit-based flow control for ATM networks. In *ACM SIGCOMM*.

[25] Lynch, B. 1968. Reliable full duplex transmission over half-duplex telephone lines. *Communications of the ACM* 11(6). Alternating Bit Protocol; End-of-Segment marker.

[26] Metcalfe, R. M., and Boggs, D. R. 1976. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM* 19(7):395–404. Section 7.2.2 introduces the end-dally as the bilateral closure of the EFTP file-transfer completion protocol.

[27] Metcalfe, R. M. 1978. Ethernet briefing. Recorded videoconference briefing, Xerox PARC. Transcript section 1:44:26–1:45:31 describes the EFTP dally and the role of the END REPLY echo in achieving bilateral assurance.

[28] Metcalfe, R. M. 2026. Personal communication on the end-dally protocol. Email, 10 March 2026. “As I recall, the end-dally was intended to speed things up. Like the NAK does in general.”.

[29] Milner, R. 1980. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag.

[30] Montazeri, B.; Geng, Y.; Alizadeh, M.; and Ousterhout, J. 2018. Homa: A receiver-driven low-latency transport protocol using network priorities. In *ACM SIGCOMM*.

[31] Open Compute Project, Open Æthernet Working Group. 2026. Open Æthernet specification, version 0.7a. OCP wiki.

[32] Ramakrishnan, K. K.; Floyd, S.; and Black, D. 2001. The addition of explicit congestion notification (ECN) to IP. *RFC 3168*.