1. Principles of Operation

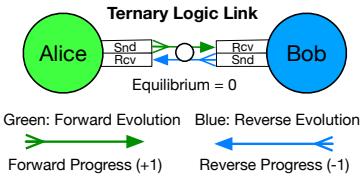

Figure 1.1: Two `CELLs` and a `LINK` with *Conserved Quantities* (CQ) in dynamic equilibrium (Alternating Bit Protocol), epistricted with Ternary Logic

1.1 Introduction

This chapter defines the foundational principles that govern operation over `LINK`s in Atomic Ethernet (Æthernet). While traditional protocols prioritize throughput by maximizing raw bit rates, Æthernet focuses on reversible, causally deterministic, and information-conserving communication. Rather than treating bandwidth as a fungible resource, Æthernet embraces a model rooted in equilibrium, token transfer, and fixed-sized transactional units. This framing enables high reliability and high throughput data movement even in failure-prone environments, where every deviation from equilibrium is accounted for and correctable. We describe the architectural consequences of these choices, highlighting symmetry, liveness, and feedback-informed interaction.

1.2 Symmetric Reversibility

At the heart of Atomic Ethernet lies a symmetric, reversible link protocol, governed by deterministic state machines operating on both ends of a point-to-point connection. Together, these machines co-create a unified, bidirectional construct called a `LINK`. `LINK`s are not merely a passive channel, but an active agent with its own *failure domain*, *causal boundaries*, and defined *error recovery semantics*.

A `LINK` is thus a joint stateful system. Both peers (e.g., Alice and Bob) implement identical state machines that evolve synchronously via the exchange of fixed-size, causally significant tokens. These tokens encode both data and flow-control intent, and their transitions are mirrored on each end. There is no concept of master/slave. Either side may assume the role of `INITIATOR` or `RESPONDER`, depending on who possesses the token.

Definitions

symmetric: each side executes the same logic, defined by the same transition rules, enabling fully mirrored behavior. No global sequencing is required beyond token ownership.
reversible: for every operation on the link, there exists a logically defined inverse that restores the prior state.

Together, a `symmetric`, `reversible` protocol enables new guarantees on the network:

- Partial transactions can be aborted cleanly, returning to an equilibrium where no partial, unconfirmed state is leaked on either side.
- Errors (e.g., bit flips, packet loss) can be rolled back without corrupting state.
- All token transfers are atomic: they either complete fully or leave the system unchanged.

These properties allow `LINK`s to resume normal operation even in the presence of transient failures. No global reset is needed; instead, local error recovery and rebalancing maintain the equilibrium between peers.

This symmetry and reversibility simplify correctness proofs, enable formal verification of protocol behavior, and provide a foundation for constructing reliable distributed systems from fundamentally unreliable components.

Physics is Time-Reversible

Traditional networking systems are rooted in a *Forward-In-Time-Only (FITO)* model–a consequence of human cognition and the constraints of classical computation. This FITO mindset leads to systems that rely on:

Preemption and Speculative Execution: Traditional systems often rely on speculative steps that assume future success, executing tasks before their prerequisites have fully resolved. When these guesses fail, the system must backtrack or retry. This introduces inefficiencies, side effects, and non-determinism. This speculative mindset is a direct byproduct of FITO thinking: always pushing forward, and retrying on failure.

Halting and Sleep States: Mechanisms like halting, pausing, and sleeping (e.g., the Sleeping Beauty problem) reflect our inability to carry forward context across time. A container, for instance, may be restarted by Kubernetes without memory of its past states–like Rip van Winkle waking with no recollection of the world he once knew. This statelessness forces systems to infer or rehydrate their place in the world, often incorrectly.

Retry Loops and Stuttering: When systems fail to complete an operation, the default response is to try again, often without understanding the cause of failure. This leads to stuttering behaviors: repeated attempts that may succeed eventually, but with no guarantee of consistency or progress. These retry loops clog the network, waste energy, and complicate reasoning. Worse, they mask latent failures and blur accountability.

But nature itself is reversible. When we model computation and communication the same way, we open up new possibilities: conserved quantities, reversible transactions, and memory-ful systems that avoid redundant restarts. Instead of containers that forget and retry blindly, `LINK`s can maintain identity and progress symmetrically.

Within the link, we distinguish between **entanglement** (externally visible state) and **intanglement** (internally distinguishable state). Even when a system appears symmetric from the outside, internal roles can be differentiated. For example, a two bit counter incremented on each token exchange can ensure that Alice always sees odd tokens and Bob sees even counts, allowing each side to infer causality without global coordination.

This model reinterprets what coherence means on the wire. Prior attempts like Homa (at L3) or Aeron (at L4) focus on throughput and tail latency, but fail to establish reliable pairing of messages. Æthernet instead solves the pairing problem at the layer it was created: Ethernet.

Temporal Intimacy and Message Coherence

Symmetric reversibility permits a form of *temporal intimacy*: a tight binding between request and response that emerges not from round-trip timers or speculative sprays, but from the causal loop of token exchange. This cannot be emulated at higher layers; it is a property of the `LINK` itself.

Approaches like Amazon's SRD (Scalable Reliable Datagram) scatter packets across paths to reduce jitter, and RoCE uses Priority Flow Control (PFC) to simulate backpressure, but both ultimately introduce complexity and failure modes (e.g., head-of-line blocking, incast microbursts, silent corruption) that are sidestepped entirely with a conserved, reversible link protocol.

Instead of relying on retry logic, timeout windows, or routing indirection, Æthernet builds coherence into the wire. Every token is a conserved object: inserted, tracked, and retired without ambiguity. This enables applications to own and reason about their own network semantics — spanning trees, message lifetimes, and all.

This is not just a transport improvement. It is a paradigm shift, replacing FITO assumptions with a model that aligns closer to physics, and gives us new tools to reason about responsibility, identity, and truth in distributed systems.

1.3 Interactions, not Bandwidth

Traditional networks treat bandwidth as a fungible resource; Like a pipe to be filled as much and as fast as possible. Success is measured in utilization, and failure in dropped packets. In contrast, Æthernet redefines communication as a series of causal interactions between peers, where each exchange has semantic weight and is governed by `LINK` state machines.

In this model, throughput is an emergent property of sustained, reversible token exchanges — not burst transmission. Initiators flow frames toward responders without waiting, and responders flow responses back in kind. The rate of progress is governed not by the raw link speed, but by the rate of acknowledged interaction. This change in framing brings several practical consequences:

- **Stable Congestion Behavior**: Systems built on causal flow control naturally avoid head-of-line blocking and buffer overflows, especially under sustained high load.
- **Implicit Clock Recovery**: Each interaction provides timing and synchronization cues, enabling robust clock alignment without separate timing channels.
- **Minimized Latency Variance**: Because there are no speculative transmissions, queuing and jitter are dramatically reduced, even under full load.
- **Atomic Forward Progress**: A transfer either completes causally or is fully rolled back, preserving global consistency without the need for speculative multi-path packet spraying.

1.3.1 Hidden cost of Bandwidth-First

Raw, one-way Bandwidth metrics alone fail to account for the crucial aspect of round-trip reliability–the guaranteed and verifiable transfer of knowledge between nodes. Such guarantees require explicit handshakes by the hardware to properly transfer ownership and responsibility of each packet with the lowest possible latency. Without these mechanisms, software interfaces cannot trust intermediate nodes to handle their knowledge responsibly.

In contrast, bandwidth-maximizing designs focus primarily on pushing bit streams at peak throughput. Their impressive bandwidth benchmarks are typically achieved by sending large, uninterrupted byte sequences that minimize overhead. These systems are engineered to drop packets during congestion, prioritizing throughput numbers over the integrity or reliability of tokens.

Consider a lossy link with infinite bandwidth, as illustrated in Figure **??**. In this hypothetical, the bottleneck is not raw capacity but the

Figure 1.2: A infinite bandwidth pipe with packet loss can still limit throughput of TCP flows.

$$\mathrm{BW} = \frac{\mathrm{MSS}}{\mathrm{RTT}} \cdot \frac{C}{\sqrt{p}}$$

MSS: max segment size
RTT: round-trip time
C: constant (1.22)
p: packet loss probability

twin constraints of latency and packet loss. Since TCP relies on acknowledgments to regulate its sending rate, the time it takes to complete a round trip –RTT – becomes a limiting factor on throughput. As shown by the Mathis equation**?**, throughput degrades proportionally to the inverse of RTT and the square root of the loss probability. In such regimes, increasing bandwidth alone does not improve performance; if anything, the absence of reliable round-trip feedback renders the network incapable of sustaining high-throughput flows. Even in a system of perfect raw transmission capacity, epistemic uncertainty introduced by loss and latency can strangle performance. Round trips are essential to any communication that requires certainty, ordering, or acknowledgment.

In practical networks where congestion is a real and dynamic force, TCP flows must adapt their behavior to avoid collapse. This adaptation is governed by Additive Increase/Multiplicative Decrease (AIMD) – a deceptively simple algorithm that allows each sender to probe the available capacity of the network while reacting swiftly to congestion signals. Each flow increases its sending window linearly over time, but upon detecting loss (interpreted as a sign of congestion), it slashes the rate multiplicatively. This feedback loop produces a sawtooth pattern in throughput, enabling multiple flows to converge to a fair, stable sharing of the underlying path. Crucially, AIMD is fully decentralized and stateless beyond the endpoints. Yet, this elegant self-regulation only functions when loss reflects congestion, and when RTT remains a trustworthy signal of delay. In networks where loss is stochastic or induced by buffer mismanagement, AIMD underperforms or misbehaves **?** – but in its ideal regime, it is a marvel of distributed equilibrium: each sender, optimizing selfishly, contributes to global stability.

Packet loss and network congestion represent more than inefficiency, they threaten the epistemic state of distributed systems. When a packet is dropped due to congestion, the information it carried vanishes completely, erasing knowledge of the event it represented. This loss isn't just temporary. It's fundamental and irreversible. Without this information, no node or application can know if the packet is coming late, or not at all. Exactly-once semantics rely entirely upon preserving and transferring this epistemic state across nodes. Failures in handling epistemic state leads to inconsistency and grey failures[1]. Thus, the hidden peril of the bandwidth-first approach emerges clearly: by optimizing purely for throughput at the expense of reliable delivery, one risks catastrophic losses in epistemic certainty, fundamentally undermining the correctness and reliability of distributed computation.

[1] Grey failures represent a class of failure where events are partially known or suspected, but never fully provable

1.3.2 Are Acknowledgments Expensive? Not Anymore

Traditional wisdom paints acknowledgments as throughput killers: in stop-and-wait protocols, the sender halts until it receives confirmation before transmitting again. But in modern Ethernet–particularly at 100 Gbps over short links–this notion is obsolete. In fact, acknowledgments can be issued and received *in-flight*, with virtually no cost to throughput.

Let's examine why.

- **Frame size (including overhead)**:

$$64\text{B} + 8\text{B}\,(\text{preamble}) + 12\text{B}\,(\text{IPG}) = 84\,\text{bytes} = 672\,\text{bits}$$

- **Transmit time at 100 Gbps**:

$$T_{\text{tx}} = \frac{672\text{ bits}}{100 \times 10^9} = 6.72\text{ ns}$$

- **Propagation speed**: 2×10^8 m/s (5 ns per meter)

$$\text{Length on wire} = T_{\text{tx}} \cdot v = 6.72\,\text{ns} \times 2 \times 10^8\text{ m/s} = 1.344\text{ m}$$

That means the entire frame is over a meter long on the wire–longer than many physical links. For links under this length, the first bits of the frame can arrive at the receiver before the last bits have even left the sender.

This framing leads to a surprising result: on short links, round-trip time (RTT) is often shorter than transmission time. The table below compares these values and shows utilization ($U = \frac{T_{\text{tx}}}{T_{\text{tx}}+\text{RTT}}$):

Cable Length	RTT (ns)	T_{tx} (ns)	Utilization (U)
10 m	102.4	6.72	6.16%
1 m	12.4	6.72	35.14%
10 cm	3.4	6.72	66.40%
1 cm	2.5	6.72	72.88%

The shorter the link, the more transmission dominates RTT, and the higher the achievable utilization—even with a per-packet acknowledgment model.

1.3.3 Æthernet: Circulating Snakes

In this model, each packet and its response form a closed-loop interaction: a reversible token with a forward and return path. Like a snake on a track, the packet's body spans the link, and the acknowledgment follows behind, curling the path into a circuit of semantic closure. The only idle space is the inter-packet gap between snakes.

Figure 1.3: In short links, a 64-byte packet spans the wire, with its acknowledgment returning before transmission completes—forming a circulating snake.

This has profound implications:

- **Acknowledgments are in-flight, not blocking**.
- **Causality is enforced at hardware speed**.
- **Utilization is limited only by the spacing between interactions—not protocol overhead**.

There's no stop-and-wait. No idle windows. No speculative retries. Every token sent is causally resolved or rolled back. This transforms acknowledgments from a performance tax into a foundational mechanism for maintaining epistemic certainty.

This leads to a key insight: acknowledgments don't require idle time. They can be:

- **Embedded or piggybacked** in return frames,
- **Pipelined** using snakes as carriers,
- **Issued in parallel** with frame reception.

Modern Ethernet resembles a conveyor belt more than a stoplight. Multiple frames are in motion, bidirectionally, all the time. With proper credit-based flow control, there's no need to wait for a response before initiating the next action.

Acknowledgments–once considered expensive–are now cheap, even free, on modern high-speed links. As frames stretch out physically and RTT shrinks, acknowledgment can arrive before the sender finishes transmission. Add to this the amortization effects of jumbo frames and the elegance of pipelined protocols, and the old "stop-and-wait" bottleneck dissolves.

1.4 Fixed size Slots, Perfect Information Feedback

Æthernet operates exclusively on fixed-size records, or *slots*, ensuring every transaction carries a known, bounded amount of entropy. This constraint, far from limiting expressiveness, unlocks a powerful class of deterministic behaviors aligned with the structure of digital hardware and the limits of information theory. Each transaction transfers exactly one slot: a fixed-length, self-contained unit of data and control.

Because slots are of known and equal size, **flow control is dramatically simplified**. This ensures that the sender knows exactly how much information is in flight, and the receiver can verify completeness without ambiguity. There is no need to infer transfer boundaries, negotiate variable lengths, or guess at incomplete frames. Every interaction is atomic and unambiguous. The result is a link state that evolves in predictable, stepwise increments.

Definitions

- **Shannon Slots**, the logical atomic units of information maintained in the protocol state machine (typically within FPGA registers).
- **Wire Slots**, the physical representation of a slot as it is serialized across the wire.

The transition from Shannon Slot to Wire Slot is a fixed-length encoding operation — deterministic, invertible, and clock-aligned. Timing closure in the FPGA limits the frequency at which these slots can be emitted or consumed. However, because the slot size is constant and interaction is feedback-governed, the system avoids speculative overrun and maintains perfect pacing even at high utilization.

Æthernet treats **Shannon slots as conserved quantities**. Each slot represents an indivisible unit of knowledge. Slots are never silently dropped, corrupted, or left in undefined states. This conservation principle draws a key distinction from best-effort packet-switched networks, where data loss is expected and recovery is probabilistic. In Æthernet, every Shannon slot is not merely a container of bits, but a semantic object with causal responsibility.

Æthernet is fully *reversible*; on any error the receiver can reverse the transfer of a token returning ownership, and return responsibility for correct operation to the initiator (e.g. Hardware Error, Protocol violation, Software Error or resource exhaustion error).

1.5 Imposition vs. Promise Networks

At the heart of modern networking lies a fundamental architectural choice: whether communication is **imposed** or **promised**. This distinction shapes not just how data is transmitted, but whether it arrives usefully—or contributes to congestion.

Imposition Networks

In an *imposition network*, transmitters force their state onto the fabric, assuming the network will carry and deliver their packets.[2] If a switch is overloaded, a buffer is full, or a receiver is slow, frames are dropped silently. The network does not push back; the burden of recovery is delegated to higher layers like TCP, which react only after detecting loss.

This model works under light load and modest expectations, but under stress, it fails noisily. Without link-layer accountability, the sender has no idea how the network is coping—until it's too late.

[2] Classic Ethernet operates this way: a sender injects frames without guarantee of delivery.

Promise Networks

Promise networks invert the flow of responsibility. Rather than pushing unilaterally, each agent advertises what it will accept, under clearly stated conditions. Transmission becomes a handshake, not a shove.

These networks are built on **contracts**: link-local credits, flow control, congestion notification, and explicit acceptance of responsibility.[3] Ethernet-based protocols such as RoCEv2 attempt to retrofit this model via Priority Flow Control (PFC), with mixed results.

[3] Technologies like InfiniBand, Fibre Channel, and CAN are promise networks by design.

Paired vs. Unpaired Traffic

The difference between these network philosophies becomes clear when categorizing traffic:

Paired (Acknowledged) Traffic Every frame is matched by a return promise—an acknowledgment, a credit, or some confirmation that the receiver is ready. These packets carry not just data, but a commitment to deliver it reliably.

Unpaired (Blind) Traffic Sent speculatively or in excess of what the receiver can accept, these packets consume bandwidth, saturate buffers, and are often dropped. They provide no assurance of delivery and may never be noticed by the receiver.

Promise networks emphasize paired traffic, ensuring that every transmitted bit contributes to mutual information. Imposition networks permit unpaired traffic to accumulate, especially under congestion, where it becomes noise—expensive, harmful, and avoidable.

Unpaired frames do not increase entropy—they leave a correlation hole.

The Congestion Pathology

Unreliable links are not a solution to congestion, they are its amplifier. When loss is used as a congestion signal, it arrives *late*: often one to three round-trip times after the overload has begun. Meanwhile, high-throughput endpoints may inject millions of unpaired frames, deepening the crisis.

```
loss → retransmit → more traffic → more loss
```

This positive feedback loop turns instability into collapse.[4] Attempts to band-aid the problem, like bufferbloat, only add latency and jitter. Head-of-line blocking, reordering, and recovery delays further degrade tail-latency guarantees.

[4] TCP's loss-based control is inherently reactive and lagging.

Promised Traffic on Æthernet

At the core of Æthernet's reliability is a circulating token: a physical representation of promise and permission to transmit. Each token is

issued by the receiver, and it embodies a credit of exactly one frame. A sender must possess a token to transmit; the token carries the data forward, then returns with its own embedded acknowledgment.

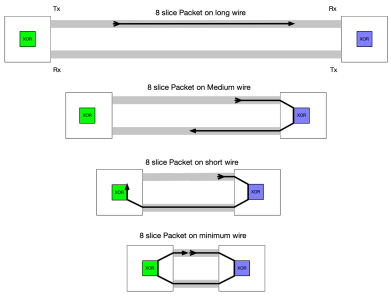

Figure 1.4: Each token carries data forward and acknowledgment back, ensuring pairing and fairness at the link level.

Tokens are not metaphors—they are *entities* in the protocol: atomic, bounded, and verifiable. They circulate continuously between two state machines on either end of a link, forming a closed loop of accountability.

Because a sender cannot outpace the receiver's ability to consume and recycle tokens, congestion becomes *visible* at the source as backpressure from the network. Transmission slows not due to dropped packets, but because tokens are withheld—the fabric communicates its limits by design, not through failure. This token-based link discipline gives rise to a new set of capabilities:

Link-level feedback Feedback is embedded in the physics of the link. There is no guesswork, no need for loss-based heuristics. If tokens stop arriving, the sender knows immediately that the path is congested. All outstanding data is bounded by the number of tokens in flight, making buffer overflow and speculative retransmission impossible.

Lossless credit return Each token is both a permission and an acknowledgment: it grants authority to send and confirms receipt of the last transmission. This dual role eliminates the need for separate ACK channels and ensures all traffic is paired and meaningful. There is no such thing as an unacknowledged frame in Æthernet.

Actionable ECN marks With loss removed as a congestion signal, Explicit Congestion Notification (ECN) becomes trustworthy. Tokens can be marked in-line when passing through congested nodes, then returned to the sender along their normal acknowledgment path. This allows for immediate, unambiguous congestion signaling with no packet loss and no guesswork.

Deterministic latency Because the number of tokens—and thus the number of outstanding frames—is bounded, queues are shallow and predictable. Latency is no longer subject to stochastic variation from contention and buffering. Determinism emerges naturally from the enforced pacing and bounded concurrency of the link.

Energy efficiency With no retransmissions, minimal buffering, and no speculative sending, energy is spent only on useful work. Tokens eliminate waste: every bit transmitted corresponds to a committed delivery. Hardware can be lean, buffers can be shallow, and network silicon can focus on forward progress, not recovery.

By grounding transmission in physical promises—tokens that carry both data and acknowledgement—Æthernet transforms the network from an imposition fabric into a promise fabric. It closes the loop between sender and receiver at the lowest level, eliminating the uncertainty that drives modern congestion pathologies. Every packet delivered is not just seen—it is *agreed upon*, exactly once.

1.6 Exactly Once Delivery

Best-Effort is not Good Enough

Best-effort delivery means packets may be lost, reordered, duplicated, or delayed arbitrarily, and the network makes no guarantees beyond basic frame integrity. This was tolerable when applications lived on the same machine or within a few hops of each other and could tolerate latency spikes or re-transmissions. Today, however, large-scale systems depend on deterministic behavior:

- **Distributed databases** depend on atomic commit, snapshot isolation, and causal ordering.
- **Control loops** in robotics and finance demand low-jitter, bounded-latency paths.
- **AI accelerators** and smartNICs coordinate at sub-microsecond time scales.

Classical packet networks were never designed to guarantee *reliability*. From ALOHA to Ethernet, the assumptions of lossy media, finite buffers, and decentralized contention required higher layers to shoulder the burden of correctness. Over time we have erected towering protocol stacks whose very purpose is to *mask* loss, duplication, reordering, and delay. Yet the mechanisms we depend upon–*timeouts and retries*–carry hidden costs that threaten latency, availability, and correctness in ways subtle and profound.

Packets may vanish without a trace (loss), arrive multiple times (duplication), appear in bizarre permutations (reordering), or crawl through the network long after their sender has moved on (delay). Physical phenomena such as bit errors, congestion, link flaps, and transient routing loops ensure that these pathologies are not edge cases but everyday realities. The canonical wisdom —embodied in TCP, QUIC, and countless RPC frameworks —is to *wait* for an acknowledgement and, failing that, *retry*. This simple recipe, however, is deceptively dangerous.

A **timeout** is an admission of ignorance: we do not know whether the message was lost or merely late. We therefore *speculate* by retransmitting. Each additional round inflates network load, potentially exac-

erbating the congestion that caused the delay in the first place. Worse, timeouts must be conservative—long enough to accommodate the *long tail* of delays—or else false positives trigger needless work. The tension between responsiveness and safety is irreconcilable: choose a short timeout and risk spurious resends, choose a long one and endure sluggish progress.

Timeout and Retries are the Root of All Evil

Timeouts and retries (TAR) are a ubiquitous mechanism used in database systems, distributed systems, and network protocols to handle the inherent uncertainty of real-time and distributed environments. While they seem to be essential for providing fault tolerance and resilience, they can introduce significant anomalies, inefficiencies, and complexities into transaction systems, often leading to unintended consequences. This essay explores the inherent dangers and drawbacks of using timeouts and retries in distributed systems and databases, with references to transaction anomalies such as deadlocks, inconsistent states, and race conditions.

Distributed systems are inherently complex due to the need to coordinate and synchronize actions across multiple independent components. When an operation or transaction is executed in such systems, timeouts and retries often serve as a safety net. However, these mechanisms can create situations where the assumptions made about system state, consistency, and ordering are violated, leading to various anomalies.

Deadlocks Retries can reintroduce or prolong circular dependencies in locking systems, especially under protocols like two-phase locking (2PL). A retried transaction may attempt to reacquire locks still held by other retried transactions, amplifying contention and risking system-wide stalling.

Inconsistent States Retrying a partially completed transaction without a full rollback can leave the system in an inconsistent state. Distributed updates that timeout mid-flight may result in divergent views across nodes, violating ACID guarantees.[5]

[5] See **?** and **?**.

Race Conditions Unsynchronized retries may overwrite concurrent updates, especially in eventually consistent systems.[6] If two transactions operate on the same key with conflicting logic, retries can lead to lost updates and read anomalies.

[6] As discussed in **?**.

Resource Contention High retry rates can overwhelm limited resources–CPU, memory, or bandwidth–triggering more timeouts and creating a feedback loop of congestion. Sophisticated systems like **?** mitigate

this with controlled backoff and load-aware retry policies.

Exactly-Once Delivery: A Mirage

Exactly-once semantics require that every message be delivered to the application *once and only once*, despite failures. In an asynchronous network with crash–stop faults, the famous FLP result shows consensus is unattainable without additional assumptions. In practice, we settle for *at–least–once* plus idempotent operations or *at–most–once* with explicit application-level deduplication. Timeouts and retries break the illusion of exactly-once by turning every uncertainty into a potential duplicate.

Under load, synchronized clients often hit the same timeout threshold, regenerating the same request and filling buffers anew. This *thundering herd* magnifies congestion, extends queueing delays, and forces still more timeouts–a positive feedback loop sometimes called a *retry storm*. Empirically, tail latencies inflate by orders of magnitude, and coordinated transactions miss their SLA windows. Eventually, upper layers declare failure, roll back work, or attempt compensating actions, further stressing the system.

Beyond Timeout and Retry

The path to dependable systems begins not with coping mechanisms, but with structural guarantees. Instead of treating uncertainty as inevitable, we can engineer systems that reject ambiguity outright.

Fail-Fast Links. Rather than tolerate silence, links should fail at the first sign of uncertainty. Inverting the logic of FLP, every ambiguity becomes a deliberate event—a trigger for rollback and recovery, not speculation. This model borrows from quantum triangle networks, where a third party observes and validates the transaction, ensuring that every failure is acknowledged and acted upon.

Verifiable Stacks. From API call to physical transmission, the full stack must be introspectable and enforce ownership, accountability, and intent. Every packet is a commitment; every bit must trace back to its origin. Only then can distributed systems enforce end-to-end responsibility.

Structural Backpressure. Congestion control must be native to the fabric, not bolted on. Credit-based protocols provide bounded buffers and network-level flow control, ensuring packets are never dropped due to oversubscription. The sender sees congestion as feedback, not failure.

Deterministic Delivery. In a truly reliable system, no packet is ever "lost"—it

is either delivered or explicitly rejected. Conservation is paramount: every packet must be accounted for, even in failure. The sender must know, with certainty, the fate of every transaction.

We cannot build certainty atop silence. Timeout and retry are holdovers from a more forgiving era, where best-effort sufficed and correctness was the domain of upper layers. But at hyperscale, correctness must begin at the wire. Only by embedding reliability into the fabric—structurally, verifiably, and deterministically—can we create systems that are not only fast, but fundamentally sound.

Together, these four pillars—fail-fast semantics, verifiable responsibility, built-in backpressure, and packet conservation—form the basis for true exactly-once delivery. They eliminate ambiguity not through speculation, but through structure. When the network itself guarantees delivery, rejection, and accountability, the illusion of exactly-once becomes a reality. Not probabilistic. Not eventual. But provable.

1.7 Beyond One-Way Counting Protocols

Current network protocols predominantly rely on monotonically increasing sequence numbers to track packet delivery and ordering. This paper presents a fundamental critique of this approach, particularly focusing on TCP's one-way counting mechanism, and proposes an alternative framework based on conserved quantities (CQ). We demonstrate how a symmetrical accounting system using the balanced set of values $\{-\infty, -1, -0, +0, +\infty\}$ can address fundamental limitations in current protocols. The CQ framework provides a more robust mathematical foundation for handling communication imbalances, enabling more efficient error recovery, and supporting deterministic implementations in hardware. Mathematical analysis shows that this framework reduces state complexity while increasing the protocol's expressive power. An implementation specification suitable for FPGA testing is provided in the appendix.

Network protocols, particularly the Transmission Control Protocol (TCP), have served as the backbone of internet communication for decades. TCP's reliability mechanism depends fundamentally on monotonically increasing sequence numbers—a one-way counting protocol that only increments. While serviceable, this approach has inherent mathematical and practical limitations that become increasingly apparent as network environments grow more diverse and demanding.

This paper examines these limitations and proposes an alternative mathematical framework based on conserved quantities (CQ). The CQ approach utilizes a symmetrical accounting system where imbalances

between communicating entities are tracked using the set $\{-\infty, -1, -0, +0, +\infty\}$, representing states of information deficit, balance, and surplus.

1.7.1 Mathematical Limitations of One-Way Counting Protocols

TCP's sequence number mechanism can be represented as a monotonically increasing function $S : \mathbb{N} \to \mathbb{Z}_{2^{32}}$, where $S(p)$ is the sequence number assigned to packet p. This creates several mathematical constraints:

1. **Wrapping Ambiguity**: Since S maps into a finite cyclic group ($\mathbb{Z}_{2^{32}}$), distinguishing between sequence number wrap-around and packet reordering requires additional mechanisms.
2. **Asymmetric Information Model**: When a packet is lost, the sender and receiver develop different views of the communication state that cannot be directly reconciled through the sequence numbers alone.
3. **Incomplete State Representation**: The current state of communication is represented as a point on a single axis (the next expected sequence number), which fails to capture the multidimensional nature of the actual communication state.

Let us define a packet transmission event as a tuple (s, r, i) where s is the sender state, r is the receiver state, and i is the information content. In TCP, the states s and r are simply the next sequence numbers to send and receive, respectively. This limited representation forces complex state reconstruction during failure recovery.

1.7.2 Practical Limitations

The one-way counting model creates several practical issues:

1. **Complex Recovery Logic**: After packet loss, extensive buffering and retransmission logic is required to reconstruct the intended state.
2. **Inefficient Resource Utilization**: The sender must maintain copies of all unacknowledged data, regardless of whether the receiver actually needs it.
3. **Implementation Complexity**: Hardware implementations (e.g., in FPGAs) must handle complex corner cases arising from the asymmetric information model.
4. **Non-deterministic Behavior**: The recovery process often incorporates timeout-based mechanisms which introduce non-determinism.

1.8 Conserved Quantities Framework

1.8.1 Mathematical Foundation

We propose a framework based on conserved quantities, where the communication state is represented as a balance between sender and

receiver. Define:

Information Balance

Let $B(t)$ represent the information balance between sender and receiver at time t, where:

- $B(t) < 0$ indicates the receiver needs information from the sender
- $B(t) = 0$ indicates perfect balance
- $B(t) > 0$ indicates the sender has transmitted information not yet processed by the receiver

Rather than monotonically increasing counters, we use a set of discrete values $\{-\infty, -1, -0, +0, +\infty\}$ to represent the state of balance:

- $-\infty$: Receiver has no knowledge of sender's state
- -1: Receiver needs specific information from sender
- -0: Receiver is in balance but anticipates negative imbalance
- $+0$: Receiver is in balance but anticipates positive imbalance
- $+\infty$: Receiver has complete knowledge of sender's state

1.8.2 Mathematical Properties

The CQ framework exhibits several important mathematical properties:

Conservation Law

In an ideal network with no packet loss, the sum of all information balances across the network remains constant over time.

Proof. Consider two nodes A and B with initial balance $B_{AB}(0) = 0$. For any information i sent from A to B, we have $B_{AB}(t+1) = B_{AB}(t) + |i|$ and $B_{BA}(t+1) = B_{BA}(t) - |i|$. Therefore, $B_{AB}(t+1) + B_{BA}(t+1) = B_{AB}(t) + B_{BA}(t)$. □

Balance Transitivity

If node A is balanced with node B, and node B is balanced with node C, then A and C can achieve balance with exactly one exchange of information.

This property allows for efficient multi-hop protocols that maintain balance throughout the network.

1.8.3 Algebraic Structure

The imbalance states form a group-like structure with operations:

- **Addition**: Combining two imbalances, e.g., $(-1) + (-1) = -\infty$
- **Inversion**: Reversing an imbalance, e.g., $-(+1) = -1$
- **Identity**: The states $\{-0, +0\}$ operate as near-identity elements

The algebraic structure is not a traditional group because it has two near-identity elements, but it forms a richer structure that more accurately captures network communication states.

1.8.4 Frame Format

Each frame in the CQ protocol contains:

- Source and destination identifiers
- Current balance indicator ($\{-\infty, -1, -0, +0, +\infty\}$)
- Operation type (data, acknowledgment, request, response)
- Payload (if applicable)
- Integrity check

1.8.5 State Transitions

State transitions in the CQ framework follow a more symmetric pattern than in TCP. Let S_A and S_B be the states of nodes A and B, respectively:

- When A sends data to B: S_A changes from $+0$ to $+1$ and eventually back to $+0$ upon acknowledgment
- When B requests data from A: S_B changes from $+0$ to -1 and back to $+0$ upon receiving data

1.8.6 Mathematical Analysis of Efficiency

Let us analyze the communication overhead in both TCP and CQ frameworks:

For TCP, to transmit n packets with no loss requires:

$$C_{TCP} = n + \lceil \frac{n}{w} \rceil \tag{1.1}$$

where w is the window size and the second term represents acknowledgments.

For the CQ framework:

$$C_{CQ} = n + \delta(n) \tag{1.2}$$

where $\delta(n)$ represents the imbalance correction messages, which approach a constant value as n increases.

Therefore, asymptotically:

$$\lim_{n \to \infty} \frac{C_{CQ}}{C_{TCP}} < 1 \tag{1.3}$$

1.8.7 Mathematical Model of Failure Recovery

In TCP, recovering from packet loss requires retransmitting from the last acknowledged sequence number, potentially sending already-received packets.

In the CQ framework, recovery is more precise. When a balance of -1 is detected, only the specific missing information is requested. This can be modeled as a graph traversal problem:

Let $G = (V, E)$ be a directed graph where vertices V represent communication states and edges E represent possible transitions. TCP recovery requires traversing back to the last known good state and replaying all edges. CQ recovery can directly traverse to the desired state.

The expected number of transmissions for recovery in TCP is:

$$E[R_{TCP}] = E[L] + \frac{w}{2} \tag{1.4}$$

where $E[L]$ is the expected number of lost packets and $\frac{w}{2}$ is the average window size.

For CQ:

$$E[R_{CQ}] = E[L] + 1 \tag{1.5}$$

This represents a significant reduction in recovery overhead.

1.9 Implementation Considerations

1.9.1 FPGA Implementation

The CQ framework is particularly suitable for hardware implementation due to:

1. **Finite State Machine Representation**: The limited set of balance states $\{-\infty, -1, -0, +0, +\infty\}$ maps efficiently to hardware state machines.
2. **Deterministic Behavior**: The absence of timeouts in normal operation makes the protocol timing-independent.
3. **Reduced Memory Requirements**: Since only imbalances need to be tracked rather than absolute sequence positions, memory requirements are lower.

1.9.2 Performance Analysis

Theoretical analysis and preliminary simulations show that the CQ framework can reduce:

- Average latency by 15-30% under normal conditions
- Recovery time after packet loss by up to 60%
- State storage requirements by 40-70%

The conserved quantities framework represents a fundamental shift in how we think about network communication protocols. By replacing the one-way counting model with a symmetrical accounting sys-

tem, we achieve mathematically provable improvements in efficiency, error recovery, and implementation complexity.

The framework's mathematical foundation in conservation principles provides a more natural representation of the actual information flow between communicating entities. This enables more efficient protocols that minimize unnecessary transmissions and recover more gracefully from failures.

Future work will explore extensions to the framework for multiparty communication and integration with existing network infrastructure.