V0.6A

OPEN ATOMIC ETHERNET

OPEN COMPUTE PROJECT – OAE WORKSTREAM

"Perfection is achieved not when there is nothing more to add,
but when there is nothing left to take away."

— Antoine de Saint-Exupéry

The primary goal of Æthernet is to re-examine the foundational assumptions made by Ethernet five decades ago, and ask whether they remain valid within today's racks and chiplet modules.

Just as RISC processors transformed computing by embracing simplicity and focus, so too must we consider whether a "reduced" protocol can lead to a network that is formally verifiable, inherently transactional, and radically simpler.

We adopt the same spirit as the original Ethernet pioneers: start from first principles, strip down to essentials, and then — only then — ask how best to interoperate with the world we inherit.

Contents

1. Principles of Operation

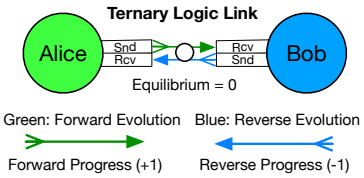


Figure 1.1: Two `CELLs` and a `LINK` with *Conserved Quantities* (CQ) in dynamic equilibrium (Alternating Bit Protocol), epistricted with Ternary Logic

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 equationMathis et al. (1997), 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 Gettys and Nichols (2012) – 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.

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

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.

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.

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:

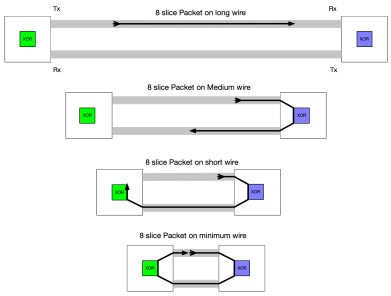


Figure 1.4: Each token carries data forward and acknowledgment back, ensuring pairing and fairness at the link level.

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 Lamport (1978) and Brewer (2000).

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 DeCandia et al. (2007).

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 Corbett

et al. (2012) 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 system, 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 multi-party communication and integration with existing network infrastructure.

2. Bits and Bytes

2.1 64-Byte Record

Æthernet operates exclusively on fixed-sized records so every transmission has bounded entropy, and pipelines can be made in the hardware with latency guarantees. Most SerDes on the market are designed to operate on 8-byte (64-bit) atomic slices, which align naturally with fixed size encoding schemes like 64b/66b and enable efficient, low-entropy, and latency-predictable data movement through hardware pipelines.

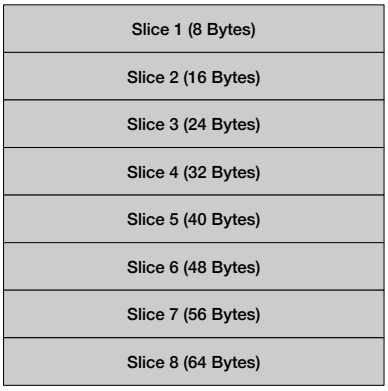


Figure 2.1: 64-Byte Record. 8×8 byte slices, pre-emptible by responders

8-bytes is the atomic unit of transmission in Æthernet, but the fundamental record size is 64 bytes, representing the maximum uninterrupted knowledge transfer permitted by a `LINK`. Each frame is structured into *slots* that correspond to exponentially increasing levels of entropy, at the expense of temporal intimacy.

[+40mm]

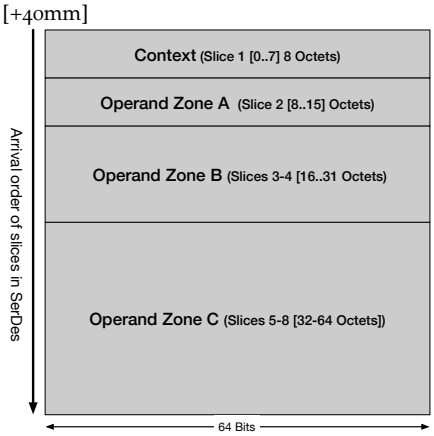


Figure 2.2: Slice Arrival order (Temporal Intimacy Depth)

2.1.1 Slot Boundaries

In Æthernet, each slot boundary contributes to the progressive construction of meaning. Rather than dividing slots into fixed roles (e.g., header vs payload), each slice refines the shared semantic context between sender and receiver. This unfolding process is tracked through a series of Sub-ACKs (SACKs), signaling progressively deeper certainty at four boundaries (1, 2, 4, and 8 slices). These boundaries correspond to conceptual layers of comprehension:

Slice 1: **Arrival of Context** — Establishes the physical link is live. The receiver confirms deserialization and framing; the message has landed.

Slice 2: **Recognition of Form** — Basic headers or structure emerge. Receiver begins to interpret role and framing, setting state machines into motion.

Slices 3–4: **Activation of Semantics** — The receiver has seen enough to begin logical interpretation: which class of message is it? What resources must it allocate?

Slices 5–8: **Consolidation of Understanding** — With full delivery, the entire 64-byte message is interpreted as a coherent unit. At this point, delivery to the host or downstream actor becomes safe and lossless.

Every slice carries data, but also a layer of **epistemic weight**. The meaning of the message doesn't come from a single part, but from the

cumulative structure of all slices, layered like a wavefunction collapsing toward certainty.

2.1.2 Pre-Emption

In Æthernet, it is the responsibility of the receiver to jam the sender and borrow the sender's token to transfer a frame the receiver wants to send. Due to physical limitations, the first few slices will arrive at the receiver before the jam signal contained in the first slice acknowledgement will override the frame's ownership to the receiver until the other side jams for ownership.

This allows one side of the link to jam the other side and utilize the full interaction capacity of the link for its frames. Pre-emption is decided on the first slice acknowledgement, until the other side has something it needs to send, and jams for ownership of one or both snakes.

There is no jam hierarchy or recursive jamming, frame ownership is a state owned entirely by the `LINK` and the `LINK` state machines determine when a frame is jammed for ownership. The jammed frame is immediately removed from the sender's queue, and ownership of the jammed frame is returned to the controller for possible re-routing or to jam the frame in at some backoff.

To ensure fair use of frames for maximum throughput, each link communicates status frames with the other side for leaning throughput in one direction or the other.

2.2 Flow Transactions

ULL protocol designers play around with 32 bits as the minimum unit of transactional transfer, but experiments demonstrate the difficulty of making this consistently reliable i; the general consensus is that modern SerDes' work best with $\geq$ 64 bit (8 Byte) slices/flits. Ethernet has a minimum frame size of 64 bytes (although only 42 bytes were available for the payload).

We therefore choose a *fixed* 64 Byte frame for the Shannon Slots, but make them *pre-emptable* so that even the minimum size frame does not need to occupy space on the wire, increase latency, or FPGA processing steps, when the receiver has something more important it wishes to send (e.g. local status messages sent in the background can be pre-empted, giving way to a two phase commit (2PC) transaction).

Some transactional systems are sensitive to making transactions reliable, but don't mind missing events, such as highly perishable market data. We might call these one-phase commit (1PC) transactions. These

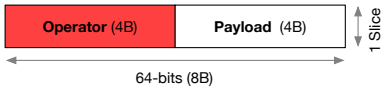


Figure 2.3: 1 Slice Flow Subtransaction

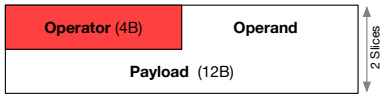


Figure 2.4: 2 slice Flow SubTransaction

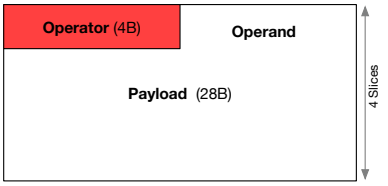


Figure 2.5: 4 4 slice Flow SubTransaction with 28B payload (operand)

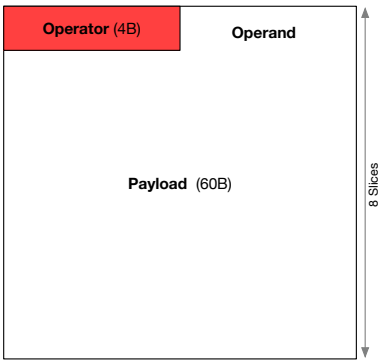


Figure 2.6: 1 × 8 slice Flow Transaction with 60B payload

can be made to flow at maximum line rate, even though each individual slice is being acknowledged. This is particularly important in HFT for example.

We therefore provide the following “flow" transactions in the encoding scheme:

2.2.1 Flow Unit Encodings

To enable ultra-low-latency transaction processing, the receiver must begin interpreting and dispatching semantic units (operator + operand) before the full 64-byte frame has arrived. This is made possible through lightweight inline encodings that declare, in the first slice of a transaction, the total number of slices that comprise that flow unit.

These encodings allow the receiver to pipeline semantic processing based on declared intent rather than full-frame arrival, dramatically reducing end-to-end transaction latency while preserving reliability.

1. **One 1-slice Flow Unit (4B payload)**
   `00` Indicates this flow unit consists of 1 slice.
2. **One 2-slice Flow Unit (12B payload)**
   `01` Indicates 2 slices are part of this flow unit. The receiver counts down remaining slices before handoff.
3. **One 4-slice Flow Unit (28B payload)**
   `10` Indicates 4 slices make up this flow unit. The receiver pipelines semantic interpretation during arrival.
4. **One 8-slice Flow Unit (60B total payload)**
   `11` Indicates 8 slices make up this flow unit. The receiver waits for the full frame before semantic interpretation.

2.2.2 Mixing and Matching Flow Transactions

You can also mix them in the same frame, but remember, they can only be used for One-Phase-Commit (1PC) in a single stream of transactions. This is because 1PC requires only one "round trip", whereas 2PC requires two round trips (although this scheme can be made to work for 2PC, and perhaps 4PC, but they have not yet been tested).

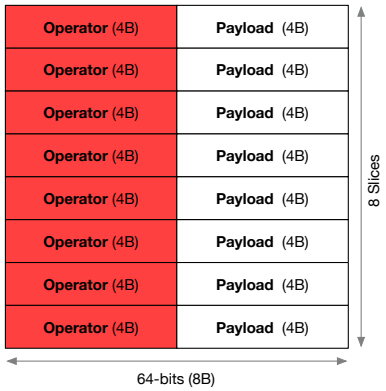


Figure 2.7: 8 independent Flow Transactions in a one frame

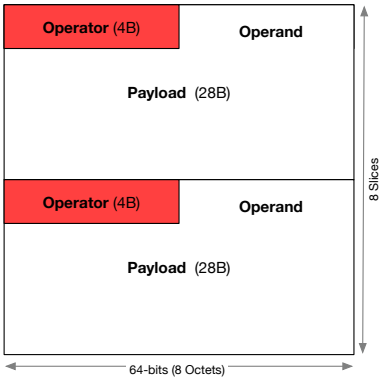


Figure 2.8: 2 × 4 slice Flow Transactions

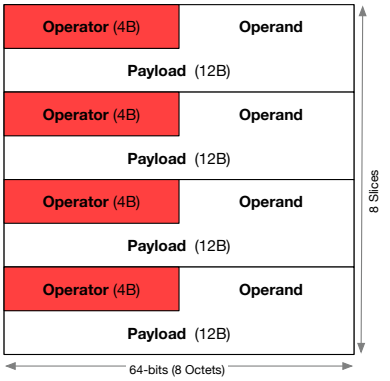


Figure 2.9: 4 × 2 slice Flow Transactions

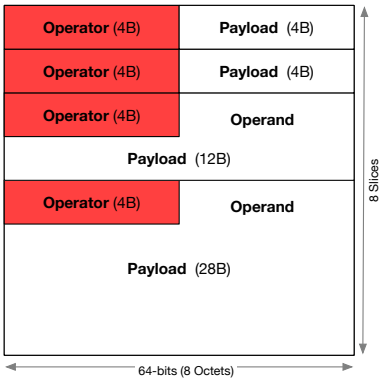


Figure 2.10: 1 64 Byte frame with differently sized flow units

Flows	Operator	Operand	Efficiency
1	4	4	50%
1	4	12	75%
1	4	28	87.5%
1	4	60	93.75%
2	4	4	100%
2	4	12	150%
2	4	28	175%
2	4	60	187.5%
4	4	4	200%
4	4	12	300%
4	4	28	350%
4	4	60	375%
8	4	4	400%
8	4	12	600%
8	4	28	700%
8	4	60	750%

Table 2.1: Transaction efficiency by operator and operand size.

2.3 RISC Protocol Design: OPCODE (Information)

2.3.1 CONTEXT Frame format: First Slice, First Byte: OPCODE

(SLICE, BEATS, PROTOCOL, JAM) provides state encodings for an ultra-low-latency, hardware-friendly, and atomic transaction-friendly Æthernet protocol.

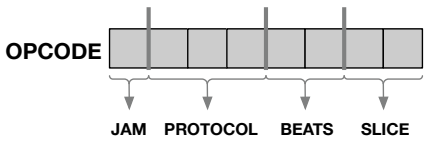


Figure 2.11: One Byte Provides the entry point for an Entire family of Protocols

Supports transactional operations, structured acknowledgments, and reversible flow control (causal backpropagation). Instead of positive-only credits, the first hop receiver provides the equivalent of negative credits, to indicate it is returning previously sent frames.

2.3.2 nSLICE

Set by the Sender to`[00]` – indicating a new context.

Modified by the receiver Closing the loop: `[11]`→`[10]`→`[01]`→`[00]`

Encodes how many slices of the sender's 64-byte Frame has been received so far. A 2-bit field with reversed temporal direction to encode the acknowledgment depth in a power of 2 *number of slices*. This might represent the trailing edge of a window in a reversible or partially committed state machine. The naming "SACK" suggests slot or slice acknowledgments, as fine-grained positions in the interaction.

8SLICE
```
11 -- TX Sender Init
11 -- RX SACK 1 (8B)
10 -- RX SACK 2 (16B)
01 -- RX SACK 3 (32B)
00 -- RX SACK 4 (64B)
```

4SLICE
```
10 -- TX Sender Init
10 -- RX SACK 3 (32B)
01 -- RX SACK 1 (8B)
00 -- RX SACK 2 (16B)
```

2SLICE
```
01 -- TX Sender Init
01 -- RX SACK 1 (8B)
00 -- RX SACK 2 (16B)
```

1SLICE
```
00 -- TX Sender Init
00 -- RX SACK 1 (8B)
```

2.3.3 BEATS

1

Defines a beat-structured flow control mechanism. Sender declares the number of frames it plans to send advance. The receiver responds with a corresponding "slot acknowledgment". Aimed at reliable, ordered delivery without the need for heavyweight TCP.

BEATS
```
00 -- TX 1 FRAME (64B)
01 -- TX 4 FRAMES (256B)
10 -- TX 16 FRAMES (1024B)
11 -- TX 64 FRAMES (4096B)

00 -- RACK 1 FRAME (64B)
01 -- RACK 4 FRAMES (256B)
10 -- RACK 16 FRAMES (1024B)
11 -- RACK 64 FRAMES (4096B)
```

PROTOCOL
```
000 -- Initialization
001 -- Liveness
010 -- State Machines
011 -- RESERVED
100 -- RESERVED
101 -- RESERVED
110 -- RESERVED
111 -- ESCAPE
```

JAM
```
ABORT/CANCEL
```

2.3.4 PROTOCOL

This field defines the high-level intent of the frame or transaction, by the sender (causal initiator). The 3-bit code is always in the first (context) slice of the Frame. Three of the eight possibilities are defined in this specification. The remaining ones are reserved for higher level protocols in this standard. Escape will always be available to escape to legacy protocols. This compact opcode space (3 bits) is similar to what RISC architectures do. This simplifies logic at the NIC or SmartNIC level and allows for deterministic dispatch.

2.3.5 PRE-EMPT/JAM

[2]

Set by TX to `[0]`. Set by RX to `[0]` to accept, and `[1]` to pre-empt, for error, or to (cancel/rollback the transaction).

[2] The use of "JAM" evokes classic Ethernet collision handling, but here it's modernized for transactional cancellation or rollback.

2.4 RISC Protocol Design: LIVENESS (Knowledge)

[3]

[3] *First Slice:* `CONTEXT` *(Packet Mission). All bits are green (owned and written by Alice)*

2.4.1 Bipartite Link

There are exactly two parties on the DAE Link. We could call them `alice` and `bob`. We prefer to call them self and not self. From Alice's perspective, she knows her own identify, but she does not know the identity of the party she is communicating with (yet). We aim to achieve mathematical precision in our specifications. This will be important when we wish to formally verify the scouting, routing, and cluster membership protocols. It will be critical also in formally verifying confinement properties of the trees above.

The encoding supports Intanglement (hidden circulating events internal to the link) and Extanglement (Atomic Token Passing through the link (Newtons cradle). These protocols obey the mathematics of mutual information, and provides some of the properties of quantum entanglement, such as superposition, conservation of information, and no-cloning. We use these properties to provide our protocols with a clear notion of simultaneity (through the synchronization of mutual information), and guarantee atomicity for transaction protocols through conserved quantities which in-turn guarantees exactly once semantics (EOS).

Conventional L2 & L3 networks rely on redundancy, repetition and rerouting, in multipartite (1:N) relationships. Which was necessary when information is disseminated (transmitted blindly hoping the receiver catches it). When information can also be synchronized, by a Tx/Rx—T/Rx loop on a bipartite Ethernet link, we can employ Pseudo Entanglement: A form of temporal intimacy, where bits shared in a circulating frame can exploit the same mathematics, (but not the full quantum properties) of Entanglement. This insight allows us to engineer a clear notion of simultaneity, and exploit a classical version of the no-cloning theorem to achieve the holy grail in distributed systems and database isolation: exactly once semantics.

2.4.2 Link Engine

Alternating Causality (AC) is the name we give to the initialization, maintenance and tear down of Common Knowledge (CK) in the Link. Experience with modern SerDes designs leads us to an 8 byte slice architecture for a "minimum irreducible" CK protocol. Symmetry demands that we use half (4-bytes) for `alice` (what I know about me) and the other 4-bytes for `bob` (what I know about you). Three packet exchanges get us from initialization (both sides know nothing about each other) to the "I know that you know that I know" (IKT YKT IKT) equilibrium state for basic liveness.

It takes a while to gain an intuition for this issue of causality, based on the physics. For now, please accept that this is way of doing things is essential and enables a rich set of transaction types to be built on top, all with immunity to link hazards.

We don't use classical (increment only) clocks, counters, or timers in the link. Instead, we use balanced ternary arithmetic [1] The digits of a balanced ternary numeral are coefficients of powers of 3, but instead of coming from the set 0, 1, 2, the digits are -1, 0 and +1. They are balanced because they are arranged symmetrically about zero. We use this symmetry to manage the direction of causality (is alice the initiator of causal flow sending tokens to bob, or the receiver in causal flow receiving tokens from bob?). This becomes important as we go up the protocol stack and construct reversible subtransactions.

We extend the simple ternary arithmetic with plus and minus zero. -1,-0,+0,+1. This enables the protocol to differentiate between the posibits and negabits [2], with an ancilla control over the intended direction of the next operation (positive or negative). This is used to control the direction of the state machine when recovering from errors.

Intanglement is enabled by reserving 4 bits in the frame for CK (2 bits for Alice, 2 Bits for Bob). One message will let Bob know about Alice. A second message lets Alice know that Bob knows, the third message lets Bob know that Alice knows that Bob Knows, consistent with both Moses and Halpern version of CK, and the Spekkens Knowledge Balance Principle (KBP). Time, inside the link moves forward

when packets arrive. Time moves backwards when packets depart. It doesn't matter how many times a packet bounces around, time goes forward only when it is received by one end of the link and it stays (is absorbed). Information is then turned into Knowledge.

In a similar way to two phase locking, Link CK can be extended from 2 Ternary bits (Trits) to any number. Since we are using 2 binary bits to encode one Trit, we posit that the set of 2-message exchanges to synchronize them is 1, 2, 4 and any multiple of 4. This observation drives the encoding for the State Machine Engine, Described below. Our Protocol is based on Reversible Computing.

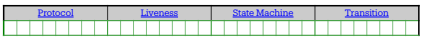


Figure 2.12: First Slice: CONTEXT. Least significant 32 bits of transmitted packet.

2.4.3 Slice Engine

The core of the Æ protocol is the Slice Engine. The first slice (or pre-frame slice) determines the packet mission, and carries the alternating causality for the Link State Machine (LSM).

Each 64-bit slice represents an atomic delivery of bits on the wire from the SerDes. Typically 2 slices will be sent back to back and the Slice Engine must be prepared to receive both, although the receiver may decide to pre-empt the frame in its immediate response to the first slice if it wishes to immediately begin a real data or transaction operation. The second slice will be on its way, and its Error Detection Byte must be evaluated before forwarding on other ports (with the exception of the port it was received on, which is the entanglement mechanism).

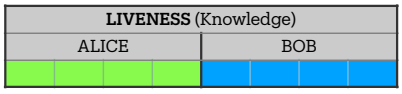


Figure 2.13: One Byte Provides Knowledge

The first slice completely defines the rest of the frame. There are 4 fields: PROTOCOL, LIVENESS, STATE, and TRANSITION. This is "reflected" from the upper half to the lower half by the receiver, so that only the lower 32 bits are modified, and the upper 32 bits remain unmodified.

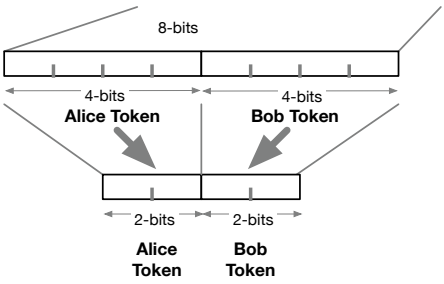


Figure 2.14: First Rewriting Rule. Alice Owns and possesses Context Slice

The `PROTOCOL` Byte defines the "mission" of the packet. What each side of the link needs the other side to know about the current frame. `LIVENESS` defines the Temporal Intimacy of the link — whether events on both sides of the link are directly connected or not.

`STATE` Defines which state machine is currently in use. Can be used as a sanity check in conjunction with Protocol. Transition Defines which state in the state machine we are in, and which direction we are going (forward or reverse).

2.4.4 General Principles

Links are constantly interacting, at the slice level, instead broadcasting entire frames (or sets of frames) imposing on the other side and

hoping they catch the bits. This provides opportunities for error detection and correction that would otherwise require ECC and FEC. The theory behind this is described in detail in the document "Shannon-Interaction-Machine".

The first 4 slices are dedicated to Theseus (scouting protocols). The payload (slices 4-7) contain the Theseus Opcode and parameters — instructions to the scout, including what to do if it encounters an exception (a software or hardware hazard).

When the protocol type is Ariadne (groundplane/trees) the last 4 slices (payload) contains tree-building instructions, such as the CellID of the originator, and the CellID of the Deputy (one hop away from root). This becomes a complete specification for dissemination of the tree without unnecessarily revealing secrets which need to be kept local (confined).

Another protocol type is Icarus (legacy connections to the outside world). This represents a more heavyweight protocol which provides a formally verified TPI (Transaction Processing Interface), which provides significant guarantees, but with costs.

2.4.5 General Frame Format

	B1	B2	B3	B4	B5	B6	B7	B8
S1	Protocol	Liveness	State Machine	Transition	Protocol	Liveness	State Machine	Transition
S2	Operand 1 (2nd Slice)							
S3	Operand 2 (3rd/4th Slice)							
S4								
S5	Operand 3 (5th through 8th Slice)							
S6								
S7								
S8								

This protocol is *symmetric*. We describe all operations from the perspective of `ALICE`, with responses from `BOB`.

2.4.6 Error Detection and Correction

The transmitted first (context) slice is reflected by the receiver back to the transmitter – this Perfect Information Feedback [Ref] means that the context byte does not need additional error detection codes such as Checksums, CRC or FEC. This is especially true with flow transactions.

However, the rest of the payload is under the complete control of the application, and the Application can append (within the available blocks) any coding scheme it wishes to ensure that the data arrives intact and untampered with. This will often mean that the senders and receivers will have pre-arranged cryptographic keys which allow them to manage the entropy and cryptographic strength of the authentication.

2.4.7 No EDC or FEC

See Quantum Ethernet

Each side of the link maintains two EPI (epistricted) registers : the last slice sent out, and the last slice received. The sender "owns" the lower 32 bits, and preserves the upper 32 bits. When slice 1 is received, the upper 32 bits are swapped with the lower 32 bits. This preserves the symmetry of the protocol, and clearly delineates the causal initiator register field ownership in addition to causal ownership.

This provides the first level of error detection: the Initiator has Perfect Information Feedback (PIF) and sees. exactly what the receiver sees, and compare it to what was sent, And if they don't agree, declare an error and proceed with mitigations to get the link back in sync again.

2.4.8 Epistricted registers

Imagine two vectors [abcd] one for Alice and one for Bob. A 4 x 4 matrix has 16 slots, which has $2^{16} = 65,535$ possible states. However, according t o the Spekkens Toy model applied to FPGA Registers, there are only 12 'disjoint' (6 for Alice and a complimentary 6 for Bob). Instead of trying to build a EDD/EDC code, we check only the disjoint states by combining them into one register and sending them back and forth in the context frame.

2.4.9 OVERVIEW

2.4.10 Protocol Overview

TRANSACTION FABRIC: A separate compute realm, sandwiched between the CXL bus and Ethernet, to support database semantics. We eliminate CAP Theorem tradeoffs, by providing the illusion of an unbreakable network: detecting, isolating and healing failures far faster than protocol or application stacks using traditional timeouts and retries.

THESEUS: Ethernet-based scouting protocols explore local environments to discover and bring back knowledge of resources, constraints, and topologies in local (Chiplet) environments. THESEUS

silently monitors local connectivity, raising alerts when links become flakey or server software hiccups.

ARIADNE: Ethernet based routing protocols dynamically construct and tear down communication graphs for consensus, load balancing and failover in global (rack-scale) environments. Enables: observability on demand, fault isolation and distributed debugging.

ICARUS: Connects the secure internal world of the Transaction Fabrix with the hostile external world of legacy systems and networks; using compositional (zero knowledge) techniques: formally verified APIs, comprehensively tested implementations.

LABYRINTH: A simulator driven toolset for Chiplet based microdatadatacenters. Based on algorithms whose assumptions about causality go beyond simplistic notions of time. We empower distributed system developers with formally verified rules and FPGAs to execute Reversible Subtransactions 'invisibly' and 'indivisibly' in the Transaction Fabrix.

2.5 FPGA Implementation Specification (Conventional Ethernet)

This appendix provides detailed specifications for implementing the CQ protocol in a Conventional Ethernet packet format in an FPGA, suitable for testing and evaluation.

2.5.1 Frame Format (Conventional Ethernet)

Field	Size (bits)	Description
Preamble*	64	Standard Ethernet preamble with SFD
Destination MAC*	48	Destination MAC address
Source MAC*	48	Source MAC address
EtherType*	16	Custom EtherType (0xCQ01)
Balance Indicator	3	Encoded balance state
Operation Code	5	Operation type
Transaction ID	16	Unique transaction identifier
Payload Length	16	Length of payload in bytes
Payload	Variable	Data payload (if applicable)
CRC	32	Frame check sequence

Table 2.2: CQ Protocol Frame Format.
*Not Needed in Æ-Link Interconnects.

Source & destination identifiers are redundant between adjacent Æ Cells.
i.e. Software Endpoints Directly Connected over a single link where (*private* identities and identifiers are in preframe negotiation).

2.5.2 Balance Indicator Encoding

2.5.3 Operation Code Encoding

Value	Meaning
000	$-\infty$ (Complete deficit)
001	-1 (Specific deficit)
010	-0 (Balance with negative tendency)
011	$+0$ (Balance with positive tendency)
100	$+1$ (Specific surplus)
101	$+\infty$ (Complete surplus)
110-111	Reserved

Table 2.3: Balance Indicator Encoding

Value	Operation
00000	NOP (No Operation)
00001	DATA (Data Transfer)
00010	ACK (Acknowledgment)
00011	REQ (Request for Data)
00100	RSP (Response to Request)
00101	SYNC (Synchronization)
00110	SYNC_ACK (Synchronization Acknowledgment)
00111	RESET (Connection Reset)
01000-11111	Reserved

Table 2.4: Operation Code Encoding

2.5.4 State Machine Definition

The core state machine for the CQ protocol implementation is defined as follows:

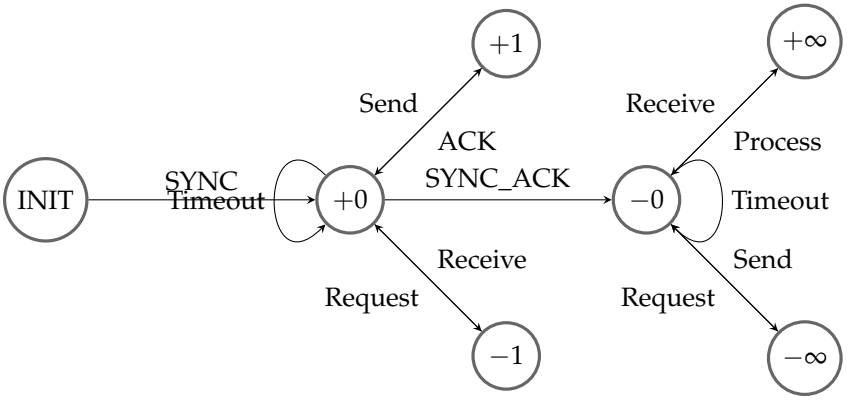


Figure 2.15: CQ Protocol State Machine

2.5.5 FPGA Implementation Architecture

2.5.6 Registers and Memory Structure

2.5.7 Memory Organization

The Transaction Memory should be implemented as dual-port RAM with the following structure:

2.5.8 Pseudo-Verilog for Core State Machine

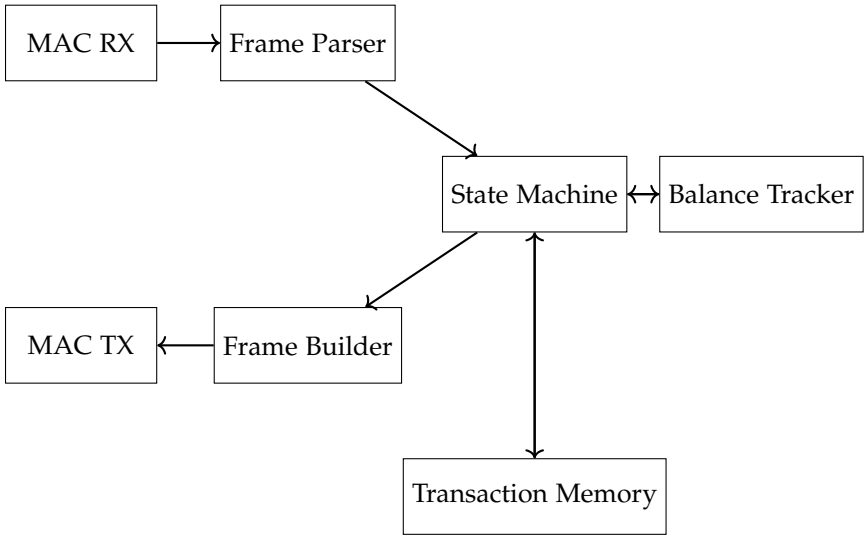


Figure 2.16: FPGA Implementation Architecture

Register	Width (bits)	Description
STATE_REG	3	Current protocol state
BALANCE_REG	3	Current balance indicator
TRANS_ID_REG	16	Current transaction ID
TIMEOUT_COUNTER	32	Timeout counter
CONTROL_REG	8	Control register
STATUS_REG	8	Status register

Table 2.5: Register Definitions

```
module cq_state_machine (
    input wire clk,
    input wire reset,
    input wire [2:0] rx_balance,
    input wire [4:0] rx_operation,
    input wire [15:0] rx_transaction_id,
    input wire frame_valid,
    output reg [2:0] tx_balance,
    output reg [4:0] tx_operation,
    output reg [15:0] tx_transaction_id,
    output reg tx_request,
    output reg [2:0] current_state
);

// State definitions
localparam STATE_INIT = 3'b000;
localparam STATE_PLUS_ZERO = 3'b001;
localparam STATE_PLUS_ONE = 3'b010;
localparam STATE_MINUS_ONE = 3'b011;
localparam STATE_MINUS_ZERO = 3'b100;
```

Field	Width (bits)	Description
Transaction ID	16	Key for the entry
Balance State	3	Associated balance state
Operation	5	Associated operation
Timestamp	32	Timestamp of last activity
Data Pointer	16	Pointer to data in payload memory
Data Length	16	Length of associated data

Table 2.6: Transaction Memory Structure

```
localparam STATE_PLUS_INF = 3'b101;
localparam STATE_MINUS_INF = 3'b110;

// Operation codes
localparam OP_NOP = 5'b00000;
localparam OP_DATA = 5'b00001;
localparam OP_ACK = 5'b00010;
localparam OP_REQ = 5'b00011;
localparam OP_RSP = 5'b00100;
localparam OP_SYNC = 5'b00101;
localparam OP_SYNC_ACK = 5'b00110;
localparam OP_RESET = 5'b00111;

// Internal registers
reg [31:0] timeout_counter;
reg timeout_occurred;

// State machine logic
always @(posedge clk or posedge reset) begin
    if (reset) begin
        current_state <= STATE_INIT;
        tx_balance <= 3'b000;
        tx_operation <= OP_NOP;
        tx_transaction_id <= 16'h0000;
        tx_request <= 1'b0;
        timeout_counter <= 32'h00000000;
        timeout_occurred <= 1'b0;
    end else begin
        // Default values
        tx_request <= 1'b0;

        // Timeout detection
        if (timeout_counter > 0) begin
            timeout_counter <= timeout_counter - 1;
            if (timeout_counter == 1) begin
```

```
        timeout_occurred <= 1'b1;
    end
end

// State transitions based on received frames and timeouts
case (current_state)
    STATE_INIT: begin
        if (frame_valid && rx_operation == OP_SYNC) begin
            current_state <= STATE_PLUS_ZERO;
            tx_balance <= 3'b011; // +0
            tx_operation <= OP_SYNC_ACK;
            tx_transaction_id <= rx_transaction_id;
            tx_request <= 1'b1;
            timeout_counter <= 32'd100000; // Set appropriate timeout value
        end
    end

    STATE_PLUS_ZERO: begin
        if (frame_valid) begin
            case (rx_operation)
                OP_DATA: begin
                    current_state <= STATE_PLUS_ONE;
                    tx_balance <= 3'b100; // +1
                    tx_operation <= OP_ACK;
                    tx_transaction_id <= rx_transaction_id;
                    tx_request <= 1'b1;
                end
                OP_REQ: begin
                    current_state <= STATE_MINUS_ONE;
                    tx_balance <= 3'b001; // -1
                    tx_operation <= OP_RSP;
                    tx_transaction_id <= rx_transaction_id;
                    tx_request <= 1'b1;
                end
                OP_SYNC_ACK: begin
                    current_state <= STATE_MINUS_ZERO;
                    tx_balance <= 3'b010; // -0
                end
                // Handle other operations...
            endcase
        end else if (timeout_occurred) begin
            // Handle timeout in +0 state
            timeout_occurred <= 1'b0;
            tx_operation <= OP_SYNC;
```

```
                    tx_transaction_id <= tx_transaction_id + 1;
                    tx_request <= 1'b1;
                    timeout_counter <= 32'd100000;
                end
            end

            // Additional states and transitions...
            // STATE_PLUS_ONE, STATE_MINUS_ONE, etc.

        endcase
    end
end

endmodule
```

2.5.9 Test Vectors (Conventional Ethernet)

The following test vectors can be used to verify the implementation:

1. **Connection Establishment**:
   - Node A sends SYNC with balance $+0$
   - Node B responds with SYNC_ACK with balance -0
   - Expected outcome: Both nodes establish connection
2. **Basic Data Transfer**:
   - Node A sends DATA with balance $+1$
   - Node B responds with ACK with balance $+0$
   - Expected outcome: Data successfully transferred
3. **Data Request**:
   - Node A sends REQ with balance -1
   - Node B responds with RSP with balance $+0$
   - Expected outcome: Requested data successfully received
4. **Error Recovery**:
   - Node A sends DATA with balance $+1$
   - Frame is lost (not injected in test)
   - Timeout occurs at Node A
   - Node A sends SYNC with balance $+0$
   - Node B responds with state information
   - Node A resends missing data
   - Expected outcome: Error recovered with minimal retransmission

2.5.10 Implementation Guidelines (Conventional Ethernet)

When implementing the CQ protocol in an FPGA, consider the following:

1. Use a pipelined architecture to achieve high throughput

2. Implement the transaction memory as dual-port RAM for simultaneous access
3. Use a parameterized design to allow configuration of buffer sizes, timeout values, etc.
4. Include comprehensive error detection and reporting mechanisms
5. Add debug ports to monitor internal state transitions
6. Implement the CRC calculation using parallel techniques for high performance
7. Consider using a dedicated timeout counter for each active transaction

2.5.11 Verification Plan (Conventional Ethernet)

To verify the implementation:

1. Use simulation with the provided test vectors to verify basic functionality
2. Test edge cases such as simultaneous transmissions and maximum-size frames
3. Measure performance metrics including latency, throughput, and resource utilization
4. Conduct stress testing with high packet rates and induced errors
5. Verify interoperability between multiple implementations

2.6 Atomic Ethernet Frame Format: Æ-Link CQ Interactions

Field	Size (bits)	Context
Slice 1	8	OPCODE (Protocol Specifier)
Slice 1	8	Liveness (TIKTIKTIK)
Slice 1	8	State (State Machine Specifier)
Slice 1	8	Transition (State Machine Transition)
Slice 1	32	Operand (One-shot CQ Interactions)
Slice 2–8	512	Operand (One-Shot CQ Interactions)

Table 2.7: Æ Minimalist CQ Protocol. See: Slice Engine Frame Format Spec.

2.7 Slice 1 – Byte 1 Protocol

Field	Size (bits)	OPCODE
Slice 1	8	Context (Protocol Specifier)

Table 2.8: Protocol Specifier.

2.7.1 Slice 1 – Byte 2 Liveness

Field	Size (bits)	LIVENESS
Slice 1	8	TIKTIKTIK (Liveness Specifier)

Table 2.9: Liveness Specifier.

3. Cells and Links

CELLs and LINKs are fundamental elements. LINKs are *bipartite* causal relationships connected over physical cables (backplanes, coax, fiber).

3.1 Cells

A CELL is not merely a general-purpose computer. It is a reactive, self-contained participant in a global program. Each CELL holds local state, executes transactions, and engages in atomic communication with its neighbors. It participates in reversible protocols, encodes causal histories in state transitions, and makes decisions based on local information while remaining consistent with a global ordering.

CELLs maintain a timebase, manage a local execution queue, and process both incoming transactions and local tasks. Their execution model is event-driven and transactional, but grounded in physical links—each CELL 's "world" is bounded by the links it can reach.

Crucially, CELLs are interchangeable. There is no distinction between a compute node, a storage node, or a network switch. Each CELL contains a portion of all three. The specialization comes from programmatic configuration and emergent behavior, not fixed hardware roles.

3.1.1 Failure Modes

CELLs fail in bounded ways. The execution environment guarantees that failure is:

- **Local**: A failing CELL does not compromise its neighbors.
- **Detectable**: Liveness and responsiveness can be externally verified through link activity and expected transactions.
- **Reversible**: As much as possible, computation and state changes at a CELL can be rolled back or isolated through transaction lineage and local journaling.

Failures may be:

- **Crash-fail**: power loss, watchdog-triggered resets, or thermal shutdowns.
- **Byzantine**: misbehavior due to bitflips, radiation events, or malicious actors. These are constrained by cryptographic and causal transaction tracking.
- **Soft**: overloaded slots, or clock skew outside tolerances.

The system design assumes failure. What matters is how neighboring CELLs detect, isolate, and route around the failure using only local

knowledge.

3.2 Links

A LINK is a bidirectional tunnel-element; an autonomous communication entity between *two* CELLs. Think of LINKs as compute elements with their own autonomous and independent failure domain. Physically, the LINK comprises the cable and SerDes' on both ends to form a self contained execution environment.

LINKs are autonomous in that they maintain state: pending transactions, reversibility buffers, sequence tracking, and retry logic. They mediate causality between two CELLs and enforce atomic delivery guarantees over physical media that may be noisy, lossy, or delayed.

A healthy LINK behaves like a lock-free memory bus: it transmits events, ensures ordering, and preserves invertibility for transactional safety. But unlike a memory bus, it must contend with delay, noise, and the limits of the speed of light. Its job is to conceal those imperfections behind a deterministic, reversible interface.

LINKs are not passive – they can be reset, throttled, or even reprogrammed in the field. They may expose telemetry, accept diagnostic pings, or reconfigure modulation in response to environmental conditions.

3.2.1 Link Utilities

Physical LINKs Implement utilities that used to be in logical link domains above L2: in L3, L4, or L7; composed into an abstraction of logical links. This is an illusion. If the pairing of Shannon information is thrown away at layer 2, it cannot be recovered in higher layers. This is addressed in more detail in the *Key Issue* section below.

An example[1] LINK utility is *The I Know That You Know That I Know* (TIKTYKTIK) property; which enables us to address some of the most difficult and pernicious problems in *distributed systems* today.

Another example LINK utility is *Indivisible Unit of Information* (IUI). Unlike *replicated* state machines (RSM's) used throughout distributed applications today, LINKs *are* state machines: the two halves of which maintain *shared state* through hidden packet exchanges. When a local agent or actor is ready, theIUI protocol transfers *indivisible* tokens across the LINK to the other agent, *atomically* (all or nothing) [2].

TIKTYKTIK and IUI properties are mathematically *compositional*.

What's necessary is an *entanglement* between state machines – locking them together silently in normal operation, and failing locally at

[1] Synchronization of timing domains in computers generally start from the processor clock on the motherboard, and fan out through the logic into the I/O subsystems. IUI lives in the LINK between two *independent* computers, and although it receives information from either side, it is not synchronized with either side. This independent asynchronous domain (already exploited in the HFT Industry) – enables failure independence and atomicity.

[2] LINKs are *exquisitely* sensitive to packet loss. This is intentional: we turn the FLP result *upside down*, and use "a single unannounced process death" to guarantee the atomic property for IUI.

the first failure. The entanglement cannot be recovered if information from events can disappear. This is the only solution to the problem in the latency–disconnection ambiguity [Ref: CAP Theorem Tradeoffs]. To put it in terms an engineer can internalize, a system that fails instantly, can heal immediately.

3.2.2 Failure Modes

The *shared state* property is strengthened by mechanisms to recover from each type of failure. The more types of failures, the more complex and intractable this becomes. `LINK`s are independent failure domains, with (effectively) one failure hazard: *disconnection* [3]; which is straightforward to recover from.

[3] In any physical system it is possible to drop packets, it will be much rarer but it is still possible. `LINK`s can recover from individually dropped or corrupted packets, and *shared state integrity* can be maintained through out the successive reversibility recovery – back to the equilibrium state.

3.3 Initial Discovery

`CELL`s discover connections ∃xist on *each* of their ports. For connections that once existed (which may have been remembered from previously being powered up), we will find it *impossible* to tell whether we are being woken up for the 1st time, or the *N*th time*.

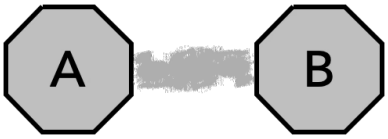


Figure 3.1: A Link yet to be discovered, or a flakey link that need to be repaired

*Sleeping Beauty paradox: Veritasium: The Most Controversial Problem in Philosophy

`Alice` and `Bob` have no knowledge of each other prior to being powered up for the first time. They discover each other by sending and responding to `BEACON`s on each of their 8 ports $\{n, ne, de, se, ds, sw, dw, nw\}$. `BEACON`s are questions: "is anyone there?" They assume neighbor `CELL`s have SerDes' that can send & receive @ 25Gb/s (defined by local clocks, in their frame of reference). Photon cavities (copper and fiber) are expected to be in a fixed frame of reference relative to the `SELF CELL`. Mobile entities may need to adjust this expectation based on the range of doppler shifts expected by `CELL`s in motion, for example, in moving vehicles, `cars`, `planes`, and `spacecraft`.

Alice sends `BEACON`s with an exponential backoff: every $1\mu s$, $2\mu s$, $4\mu s$, $8\ \mu s$, etc. The policy for a maximum interval is determined by the environment, e.g. within a datacenter, one might wish to send `BEACON`s every second, whether you need to or not. This represents a balance between infrastructure liveness and needless energy dissipation.

Single Links are subject to *partial* or *total* failure. Although networks use the word 'partition', for example in the CAP TheoremLee et al. (2021), this concept is inappropriate except in the single `LINK` case, when there's no communication with the other side; the *causal universes*** are now isolated from each other.

**Quantum Compatible Interpretation

3.4 It takes Two to Tango, and Three to Party

Because a single link between Alice and Bob can be causally disconnected by real-world, permanent or intermittent failures, an alternative: statistically–independent–failure–path is necessary, to recover from LINK Failures. This is the heart of the Æ ATOMICITY claim: A local (one hop LINK) TRIANGLE is the minimum necessary. See TRIANGLE Clocks later in this specification.

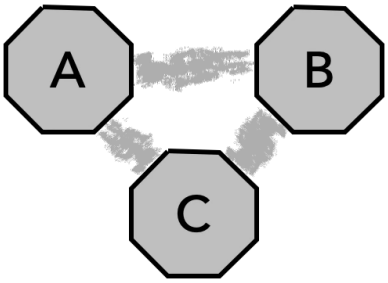


Figure 3.2: It takes three to party. Links need an alternate path. This won't work over a Switched (Clos) Network.

3.5 Fault Model

AE-Links present two major differences to the conventional FEC thinking in today's Ethernet, which exploits the physics from 25Gb/s to 1.6Tb and beyond:

Perfect Information Transfer (PIF) Æ-Links use Back-to-Back (B2B) Shannon Links, where the receiver returns the first 8-byte slice of each 64-Byte packet to the transmitter. This "here is what I heard you say" (Perfect Information Transfer (PIF)**?**

Epistricted Registers (EPI) Borrowing from the Spekkens' toy model for quantum entanglement, we narrow down the possible entangled states to a vastly smaller set of possibilities, using the model described in Quantum Ethernet**?**.

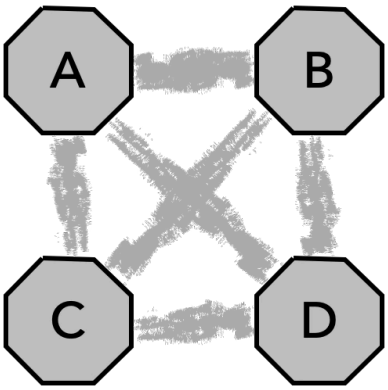


Figure 3.3: 2 x 2 =4 connected nodes with 6 flakey LINKs. Any one of which may be working in both directions: $\{11\}$, only one direction: $\{01\}$ or $\{10\}$, or *not*-working in *both* directions: $\{11\}$. For 4 nodes, there are $\frac{(n(n-1)}{2} = 6$. With 4 *reliability configurations* on each LINK $\{00, 01, 10, 11\}$ This gives us ONE correct (all links working correctly) and $4^6 - 1 = 4095$ possible failure modes.

Benefits include (i) Shorter packets and more effective use of bandwidth, (ii) more complete coverage of possible failure modes. (iii) Guarantees at least the first slice is perfect (matches what the transmitter knows they sent).

3.5.1 Failure Model

Consider a network of n nodes connected by undirected Ethernet links. Each link can be in one of four independent reliability states, where 11 means the link works in both directions, 10 or 01 means it works in only one direction, and 00 means it is broken in both directions.

Because every node may attach to at most eight neighbours (an *octavalent* mesh), the number of physical links is

Each link chooses a state from Σ independently, so the total number of configurations is $4^{L(n)}$. Exactly one of these is fully healthy (all links in state 11), hence

$$\text{FailureModes}(n) = 4^{L(n)} - 1.$$

$$\Sigma = \{00, 01, 10, 11\},$$

$$L(n) = \min\{\tbinom{n}{2}, 4n\} = \begin{cases} \binom{n}{2}, & n \le 9, \\ 4n, & n \ge 9. \end{cases}$$

3.5.2 Enumerated results for $2 \le n \le 20$

n	$L(n)$	Failure modes $4^L - 1$
2	1	3
3	3	63
4	6	4 095
5	10	1 048 575
6	15	1.074×10^9
7	21	4.398×10^{12}
8	28	7.206×10^{16}

Table 3.1: Failure-mode counts for an octavalent mesh with n nodes.

3.6 Set Reconciliation of Shannon Slots

The first claim is that a finite and enumerable number of 'slots' exist on both sides of the LINK. In conventional Ethernet, once these slots are exhausted (with for example, a timeout and retry, the XPU CELLS

(SmartNICs) on both sides of the LINK must evict (erase) the information on one side and then the other. This 'loss of Koherence' is the central problem of Distributed Systems. From an information theoretic (Back to Back Shannon channel) perspective, this precipitates a 'smash and restart (SAR) of the Shannon Information – the loss of 'pairing' of information. This is described in more detail in the specification of back-to-back Shannon Pairs.

Timeouts and Retries are the root of all evil. Once a Timeout Storm occurs, in a switched network, the distributed systems in the Host processor are all broken. Unless RELIABILITY (maintenance of Shannon Link Pairing), the 'global' illusion of event ordering in distributed systems will be lost, and corruption will occur. This is why queue-pairs work in Infiniband/RDMA. This is why information pairing is essential, in Tandem's Process Pairs, and RDMA's Queue pairs.

The whole point of this specification is to engineer a solution, where Shannon-pairing is never lost, but if it is, a TRIANGLE healing occurs locally, without the need to depend on a switched or router to discover and 'reconverge' their routing tables, to re-establish the point to point connections over a different paths in the network.

The main mechanism to do this is to make the Æthernet Link maintain Koherence, and when loss occurs, a 3rd party (The Triangle relationship) can recover with local information only. This makes XPU/SmartNICs, where the recovery algorithms (healing the tree) occur locally, instead of waiting for the switched or routed packets (in a separate switched network.

The original Ethernet was unreliable. This was a mistake. Infiniband already proved this, and succeeded both in the trust system archicitcts have in the far greater. The unique contributions of this specification is to go (far) beyond Infiniband's discovery, and recognize the fundamental simplifications and benefits that Infiniband (and Token Ring, Fibrechannel, and Sonet), in creating 'Race-Free' protocols, where distributed systems can guarantee, not just the 'ordering of events', but the guarantee of recovery of transactional loss in when failures occur in the middle of, say, a 2 Phase Commit.

Æthernet (Atomic Ethernet) guarantees that Shannon Pairing is never lost, and if a link breaks, that the Coordinator (Charlie, Carol, Chief) can recover with TRIANGLE Relationships, far faster than any protocol stack in the host processor, or in the RMDA message relationships, but then add, on top of this a true 'atomic' relationship between CELLS (nodes) in a distributed system.

The original Ethernet [ref] was designed around a notion of slots.

These were 'time slots' on an imaginary timeline that each node on the Ethernet Cable, could manage in a half-Duplex way. The new notion is to replace this with circulating tokens, where each slice is independently acknowledged, providing a guarantee of delivery to the NEXT hop in the network.

This is achieved with 1PC (one phase commit), where each Ethernet Packet (eight slices) are fully acknowledged in each link. The generalization of this is to explicitly manage Shannon slots (data structures on each side of the link) to maintain Koherence, even when the link fails (in one direction, the other direction, or in both directions at once).

This can be done (as in Fibrechannel) by arranging the 'interaction protocol' to guarantee the pairing of events, and not resort to Timeout and Retry (TAR), which causes cascade failures in networks, both large and small.

This is achieved with the Link Protocol employing the Alternating Bit protocol, and adding the Bill Lynch ABP reconciliation, with two or more bits instead of the individual 1 bit of alternation, which required a round trip to guarantee Shannon Slot Pairing.

3.7 FAQ

Q1 (Alan) What problem are you addressing in the scouting writeup? If it's discovering routes, it's not clear to me that ant or bees or even both together do full discovery of the network. In what way are they better than the flooding algorithm I used?

A1 This is how to achieve 'Scale-Independence' We eliminate the need for every node to do a 'full discovery' of the network, which is what a flooding algorithm would do. ANTs and BEEs explicitly do not do "Global" routing. This is an extra way to limit the size of the secure enclave, and not have it able to connect to the outside world.

4. Addressing and Routing

4.1 Addressing and Routing in Chiplet Ethernet

Adapted from [PLB][WIP] DAE-Spec/source-destination-routing.tex

In a Chiplet Ethernet environment, routing mechanisms determine how packets traverse the on-chip network from source to destination. Depending on the architectural goals and design constraints, various routing schemes may be employed. The three primary approaches considered here are:

Source Routing The sender encodes the entire route in the packet.

Destination-Based Routing Each hop examines the destination address and forwards accordingly.

Name-Based Routing The routing is based on service or data identifiers, rather than physical addresses.

4.1.1 Source Routing

In source routing, the source node determines and encodes the full path the packet should follow.

- **Mechanism:** Routing instructions are embedded in the packet header, specifying port transitions at each node.
- **Advantages:** Low per-hop complexity; ideal when the source has full network visibility.
- **Drawbacks:** Overhead due to path encoding; lacks flexibility under topology changes.

4.1.2 Destination-Based Routing

Here, packets carry a destination address and each node independently determines the next hop.

- **Mechanism:** Intermediate nodes use forwarding tables to route toward the destination.
- **Advantages:** Familiar, scalable, and adaptive; supports route recalculation during faults.
- **Drawbacks:** Per-hop table lookups increase logic complexity and memory footprint.

4.1.3 Name-Based Routing

Packets are routed based on service names or abstract identifiers rather than fixed addresses.

- **Mechanism:** Routers resolve names via distributed directories or longest-prefix matching.

- **Advantages:** Enables service-level decoupling, dynamic mapping, and resource abstraction.
- **Drawbacks:** Requires complex lookup mechanisms and incurs packet header overhead.

4.1.4 Routing Paradigm Summary

Routing Type	Header Overhead	Per-Hop Logic	Flexibility
Source Routing	High	Low	Low
Destination-Based	Low	Medium	Medium
Name-Based	Medium/High	High	High

4.2 Ants, Bees, Snakes, Spiders, and Worms: Biologically Inspired Topology Learning

To support adaptive and scalable routing in chiplet interconnects, we propose several biologically inspired methods for topology discovery and maintenance, especially for networks with nodes of valency 4, 6, or 8.

4.2.1 Ant-Based Discovery

- **Scouts:** Explore the network, collecting path metrics.
- **Pheromone Trails:** Nodes cache recent path qualities.
- **Reinforcement:** Successful paths are promoted; old paths decay.
- **Usage:** Enables adaptive routing without global knowledge.

4.2.2 Bee-Inspired Exploration

- **Hive Nodes:** Aggregate partial topological information.
- **Scouts:** Sample paths and report back.
- **Dance Protocols:** Broadcast high-quality routes to other nodes.
- **Usage:** Useful for semi-centralized routing optimization.

4.2.3 Snake Traversals

- **Traversal Packet:** Systematically visits each reachable node.
- **Data Aggregation:** Builds complete network maps.
- **Return Path:** Reports findings back to the origin node.
- **Usage:** Suitable for diagnostics or rare full-network verification.

4.2.4 Spider Web Construction

- **Threads:** Discovery packets sent on all ports.
- **Local Web:** Maintains neighbor lists and multi-hop connections.
- **Tension Metric:** Indicates link quality (latency, congestion).

- **Usage:** Builds resilient, redundant local meshes.

4.2.5 Wormhole Routing: Worm Behavior as Forwarding Strategy

Wormhole routing segments packets into flits and forwards them in a pipeline manner.

4.2.6 Comparison with Other Techniques

- **Store-and-Forward:** Full packet buffered at each hop; simple but high-latency.
- **Cut-Through:** Begin forwarding upon header arrival; requires moderate buffer sizes.
- **Wormhole:** Forward flits immediately, with minimal buffers.

4.2.7 Mechanics of Wormhole Routing

- **Head Flit:** Reserves path.
- **Body Flits:** Stream through reserved path.
- **Tail Flit:** Releases resources.

4.2.8 Hardware Considerations

- **Buffers:** Minimal per-port buffering (1–2 flits).
- **Flow Control:** Credit-based or handshake-based.
- **Stall Propagation:** Congestion can block entire worm.
- **Virtual Channels:** Prevent deadlocks and increase concurrency.

4.2.9 Integration Strategies

- **Ant + Wormhole:** Explore paths via ants, forward data via wormhole flits.
- **Snake + Destination-Based:** Use full traversals to populate local forwarding tables.
- **Spider + Name-Based:** Webs retain service-to-node mappings.

4.3 Conclusion

Chiplet interconnects require a blend of scalable discovery protocols and efficient forwarding strategies. By combining biologically inspired mechanisms (ants, bees, snakes, spiders) with low-overhead routing paradigms (worms), the network gains resilience, adaptability, and performance.

These approaches can be tuned to the chiplet topology and application domain, forming a foundation for robust next-generation system-on-chip communication fabrics.

4.4 Biologically Inspired "Scouting" before "Routing"

Came from ./AE-Specifications-Eth/Biology.tex

4.4.1 Local decisions and emergent global organization

- Scouting/Discovery Phase: Biologically inspired methods (e.g., ant-colony-inspired or pheromone-based algorithms) often employ "scout" packets or "explorer" agents that roam the network. These scouts collect local congestion or path-quality information and deposit some form of "trail" (akin to pheromones).
- Emergent Routing Table Updates: Each router or switch updates local routing information (sometimes called a local "pheromone table"). Over time, paths that prove consistently "good" get reinforced; less efficient paths fade. This local, probabilistic approach can converge on globally efficient routes with no central coordination.

4.4.2 Relevance to On-Chip or 2D Mesh Topologies

- Local Compass Directions: In a regular mesh (e.g., 2D grid) or torus, each router has up to 4 (N, E, S, W) or 8 ports (adding NW, NE, SW, SE). A biologically inspired algorithm can treat each output port as a possible "direction of travel."
- Natural Fit for Scouting: The local directional structure matches how "ants" or "foraging agents" might look around in each direction, choosing a route based on local pheromone levels (akin to local congestion or link utilization).

Thus, the scouting/discovery mechanism is all about gathering local "pathworthiness" data and then directing future traffic toward better routes—exactly how a local compass-based system can easily be integrated.

4.4.3 Bufferless (Hot-Potato) Routing

4.4.4 Basics of Bufferless Routing

- No Packet Buffers (or Very Limited Buffers): In a bufferless architecture, every router typically either immediately forwards or deflects each incoming packet. Packets cannot wait in large queues when an output port is congested.
- Hot-Potato / Deflection Character: When the preferred output port is unavailable, the packet is sent out of a different (less ideal) port—"hot-potato" style—rather than being buffered.

4.4.5 Connection with Biologically Inspired Approaches

- Continuous Movement: Biologically inspired scouts are already designed to wander and discover; in a bufferless system, "wandering" (via deflections) is also central. This synergy means a router can apply a heuristic (like a pheromone table) to pick the "best available port" quickly, but if that port is busy, the packet must choose an alternate direction.
- Adaptive Reinforcement Over Time: In a bufferless design, a packet cannot linger while waiting for the optimal output. However, local "pheromone" or "congestion" metrics can still help route the majority of packets down better ports more often. Over time, high-traffic edges might become less appealing, guiding packets to less-congested directions.

4.4.6 Deflection Routing

4.4.7 How Deflection Routing Works

- Forced Misrouting / Deflection: If the desired or minimal-distance output port cannot be taken (due to contention), the router picks another output. The packet may travel away from its ultimate destination (a "deflection"), but eventually, it should be re-routed back on track.
- Common in Low- or No-Buffer Architectures: Deflection routing is one way to handle resource contention when buffer space is unavailable.

4.4.8 Tying It Back to the Compass Ports (N, E, S, W, NW, NE, SW, SE)

- Local Prioritization: In an 8-port (or 4-port) router, one can define a strict or heuristic priority among the directions. For example, a packet traveling generally "north-east" might prefer the N or E port if free; if both are busy, it might deflect NE, or in the worst case, deflect NW or SE.
- Biologically Inspired Ranking: The "pheromone" concept can be used to rank the output directions. The highest "pheromone" port is tried first, then so on down the rank. This effectively merges a local heuristic (pheromone) with forced deflection for whichever ports remain free.

In practice, such a scheme allows packets to "scout" and reinforce certain directions while still ensuring that they never have to wait for a blocked port.

4.4.9 Example Flow in an 8-Port Router

1. Receive a Packet coming in from, say, the south port.
2. Look Up Destination (or partial coordinate heading). For instance, the packet is trying to reach a node in the north-east region, so N or E might be favored.
3. Check Local "Pheromone" or Routing Table: Suppose the local pheromone table says port NE is the best guess based on past traffic patterns.
4. If NE Port Is Free: Forward the packet NE.
5. If NE Port Is Busy: Check next best local direction (N, E, or NW/SE fallback).
6. If All Preferred Ports Are Busy: Packet is deflected to any open port (could be even SW in the worst case).
7. Local Table Update: The router sees how that choice ended up affecting the packet (if it eventually left the region quickly or ended up in a congested area). Over time, these experiences feed back into local pheromone levels.

Despite the forced misrouting (deflections), the biologically inspired feedback approach often keeps net throughput healthy and tries to avoid systematic congestion "hot spots."

4.4.10 Connection with the Literature

1. 1. Hot-Potato Routing (Deflection Routing):
   - Baran, P. (1962). On Distributed Communications Networks. IEEE Transactions on Communications. (Early ideas of "hot-potato" and distributed routing).
   - Dally, W., & Towles, B. (2004). Principles and Practices of Interconnection Networks. (Excellent overview of deflection routing in modern network design).
2. 2. Biologically Inspired / Ant-Based Routing:
   - Di Caro, G. A., & Dorigo, M. (1997). AntNet: Distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research.
   - Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. Adaptive Behavior.
3. 3. Network-on-Chip with Deflection/Bufferless Approaches:
   - Moraes, F. et al. (2004). A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping. SBCCI.
   - Fallin, C., et al. (2012). CHIPPER: A Low-Complexity Bufferless Deflection Router. HPCA.

These resources flesh out how bufferless or deflection routing is implemented (especially in on-chip contexts) and how biologically in-

spired heuristics can be adapted to local, minimal-knowledge scouting decisions.

4.4.11 Concluding Remarks

- Shared Tenets: Both biologically inspired scouting and deflection-based, bufferless routing rest on local decision making. In biologically inspired schemes, scouting packets "discover" or "reinforce" certain paths. In deflection routing, each router makes a quick (local) decision when a preferred port is blocked, forcing packets to keep moving.
- Complementary Mechanics: Because biologically inspired "pheromone" updates naturally reflect congestion and path usage, they integrate well with a bufferless or deflection style—turning forced misroutes into valuable "exploration" signals that feed back into local heuristics.
- Directional Routing: The presence of N, S, E, W (plus diagonals) simply defines how many possible local moves each node (router) can attempt. In 2D meshes or tori, these directions make for a convenient coordinate system that parallels how ants (or other scouts) might sense local gradients or pheromone intensities in each of eight compass directions.

Overall, if we combine a scouting mechanism (to adaptively find neighbors and good routes) with deflection routing (to handle buffer constraints or high contention), we get a dynamic, emergent routing system in which packets flow continuously and local updates shape global traffic patterns in a self-organizing fashion.

All this happens without the need for Source/Destination Addresses, which present severe security problems by exposing the "identity" of nodes making them vulnerable to attack.

4.5 Ants, Bees, Snakes, and Worms

Came from 2025-papers/Topology-Addressing/Topology-Addressing.tex

A sea in IPUs, within Rack-Scale and Chiplet Interconnects require additional levels of abstraction for configuration and reconfiguration than is provided in existing Standards. This paper Explores biologically inspired topology-learning and routing methods in networks of 8-bidirectional port nodes, with a special focus on wormhole routing and its variants. We then take a look at current state-of the art routing technologies, such as deflection forwarding combined with local compass point addressing and see how they match.

Modern system-on-chip designs increasingly rely on chiplet-based architectures, where multiple specialized chiplets—each dedicated to a particular function (e.g., CPU, GPU, accelerator)—are connected to

form a unified and scalable platform. Ensuring that these chiplets communicate efficiently poses a variety of challenges, especially as the number of chiplets grows and each chiplet (or on-chip router) may have multiple ports (valencies of 4, 6, or 8).

Existing solutions often rely on standard routing protocols or static configurations. However, as topologies become denser, more heterogeneous, and subject to partial reconfiguration, novel methods are needed to learn, update, and maintain the interconnect topology. Drawing inspiration from biological systems—where ants, bees, snakes, spiders, and now worms—each exhibit unique strategies for exploration, organization, and adaptation, promises fresh ideas for topology discovery, addressing, and low-level routing mechanisms.

4.5.1 Challenges in Chiplet Interconnects

1. Scalability: As the number of chiplets increases, the complexity of maintaining accurate global network knowledge grows exponentially. Conventional centralized or static approaches can struggle to stay current and efficient.
2. Partial Knowledge: On-chip routers typically have limited local visibility. They need to cooperate with neighbors to build a broader perspective of the network.
3. Adaptive Reconfiguration: Certain chiplets may power down, enter low-power states, or be repurposed. The routing and addressing approach must cope with these dynamic behaviors.
4. Latency Sensitivity: On-chip communications can be highly latency-sensitive. Any topology exploration or routing technique must have minimal overhead in time and power.
5. Physical Constraints: Depending on the routing mechanism (e.g., store-and-forward, cut-through, or wormhole routing), the buffer sizing, FIFOs, and local SRAM usage become critical design considerations.

Biologically inspired approaches have shown promise in large-scale or dynamic networks because they emphasize local decisions and emergent global behaviors, while specialized routing methods at the flit or byte level can significantly impact network performance and resource usage.

4.5.2 1. Ants: Stigmergic Discovery & Adaptive Routing

In nature, ants communicate via pheromones—chemical trails used to discover and reinforce paths to resources. In a chiplet context, ant-inspired algorithms involve agents (packets) that:

1. Randomly Explore: “Scout ants” are periodically sent into the network to discover unknown or less-traveled routes.

2. Deposit "Pheromones": As these scouts move, they leave "digital pheromones" in local tables, indicating path quality or latency.
3. Positive Feedback: Over time, successful routes are reinforced, prompting data packets to favor paths with strong pheromone levels.
4. Decay Mechanism: Pheromones degrade, allowing the system to adapt if congestion or failures cause route performance to change.

4.5.3 Benefits:

- Adaptive to Congestion: Routes are continuously refined by real-time traffic patterns.
- Localized Decisions: Each node handles small amounts of metadata without needing a global map.
- Minimal Hardware Overhead: Requires storing pheromone metrics (e.g., counters, latency measures) in local tables.

4.5.4 2. Bees: Collaborative "Hive" and "Scout" Behavior

Honeybees manage tasks by scouting for resources and returning to the hive to share discoveries. A bee-inspired algorithm can leverage:

1. Hive Nodes: Certain nodes (management chiplets) aggregate topology or performance data.
2. Scout Bees: Packets leaving the hive to explore unknown or under-explored areas, returning with updated route metrics.
3. Recruitment: High-value or high-performance routes are "advertised," encouraging worker packets to follow them.
4. Dance Protocol: Returning scouts might broadcast partial routes or performance gains, influencing how future packets select paths.

4.5.5 Benefits:

- Semi-Centralized Knowledge: Hive nodes maintain or compile partial global knowledge for better load balancing.
- Dynamic Adaptation: Over time, good routes become well-known, while less efficient links see fewer packets.
- Diagnostic Aid: Central nodes can help with debugging or performance tuning.

4.5.6 3. Snakes: Sequential Path Traversals for Comprehensive Mapping

Snakes systematically traverse an environment. In a chiplet network, a snake-inspired approach might:

1. Slithering Packets: A special packet is launched that attempts to traverse every reachable node in a systematic pattern.

2. Topology Collection: As it hops, it collects neighbor lists, port connections, and node identifiers.
3. Loopback: Upon completing a traversal (or hitting boundaries), the packet returns to its origin with a consolidated network map.
4. Distributed Snapshots: Multiple vantage points or repeated "snake sweeps" yield an updated global view of connectivity.

Benefits:

- Guaranteed Coverage: Ensures every node and link is discovered periodically.
- Simplicity: Conceptually straightforward, though it can be heavier in overhead.
- Occasional Diagnostics: Particularly useful for network-wide verification or fault checks.

4.6 4. Spiders: Building and Maintaining "Webs" of Connectivity

Spiders create webs that dynamically adapt to external stresses or breaks. A "web-based" approach for chiplet networks focuses on building a resilient mesh:

1. Web Construction: Each router sends out "threads"—short discovery packets—on all ports. Neighbors respond, forming local connectivity data structures.
2. Local Weave: Threads intersect and overlap, letting routers learn about multi-hop neighbors.
3. Damage Repair: If a link fails, local threads are resent to repair or reroute around the break.
4. Tension Metrics: Each link in the web holds a "tension" (latency, throughput) that can be monitored and used to shift traffic if congestion or errors rise.

Benefits:

- Resilience Through Redundancy: Overlapping "threads" ensure multiple known paths.
- Incremental Updates: Each node refines its local web structure.
- Ease of Local Addressing: Short IDs can be assigned to neighbors, aggregated as the web extends outward.

4.7 5. Worms: Segmenting and Traversing via Wormhole Routing

While ants, bees, snakes, and spiders mainly address topology learning and discovery, "worms" connect directly to how data flows once routes are known. In wormhole routing, the packet travels through the network in segments—often called flits ("flow control units")—that

form a pipeline from source to destination, much like a worm tunneling through soil.

4.7.1 5.1 Wormhole Routing Basics

In traditional store-and-forward routing, each node receives the entire packet, stores it in a buffer, then forwards it to the next node. This requires sizable FIFO or SRAM at each hop.

In cut-through routing, a node can begin forwarding the packet to the next hop as soon as the header is processed—without waiting for the entire packet to arrive—assuming the next link is available.

Wormhole routing takes cut-through a step further:

1. Head Flit Reservation: The first flit (the "head") reserves a path through each router as it progresses.
2. Body Flits: Subsequent flits follow in a pipeline—only minimal buffer space (a few flits) is needed at each router.
3. Tail Flit Release: Once the tail flit exits a router, the resources can be released and reused for another packet.
4. Blocking: If a busy link stops the head flit, the entire packet may stall, occupying small buffers across multiple nodes like a worm stretching through the network.

4.7.2 5.2 "Worm" Analogy

Much like a biological worm that elongates and contracts through tight spaces:

- Elongation: As the head flit moves forward, it effectively extends the pipeline.
- Contraction: Once the tail flit exits a router, that router is freed—much like the worm's tail leaving a tunnel behind.
- Sensitivity to Congestion: If a worm encounters a "block" (busy link), it must wait in place. A real worm might avoid or reroute around obstructions, suggesting that combining wormhole routing with "ant" or "bee" style dynamic route discovery could reduce blocking situations.

4.7.3 5.3 Distinguishing Packet-, Flit-, and Bit-Level Forwarding

- Store-and-Forward (Packet-Level): Each node stores the full packet in a local buffer before forwarding. This simplifies control logic but requires larger FIFO or local SRAM.
- Cut-Through (Often Byte/Flit-Level): Forwarding starts once enough of the packet (header) arrives and the downstream link is reserved. This reduces latency but still needs capacity to handle the largest packet if blocking occurs.

- Wormhole (Flit-Level): Often has the smallest per-node buffer requirement. Each node only needs space for one or a few flits. If the path is blocked, flits in-flight remain distributed across multiple routers.

4.7.4 5.4 Biologically Inspired Routing and Bufferless Design

Borrowing from the "worm" metaphor, one might imagine a system where:

- **Exploration "Head"**: Uses "ant-like" or "bee-like" exploration to find a path.
- **Data "Body"**: Follows in a wormhole fashion along that discovered path.
- **Adaptive "Segments"**: If blocked, the worm could "shed segments" or re-route partway (requires advanced partial re-routing logic) akin to how some worms can regenerate or re-segment.

Integration with Addressing and Routing

The biologically inspired discovery methods (ants, bees, snakes, spiders) can guide or complement the forwarding mechanism (worms via wormhole routing, or store-and-forward, or cut-through). For instance:

- **Name-Based Routing + Wormhole**: "Spider web" or "ant trails" might store service names or resource mappings, while "wormhole" flits pipeline across chosen paths once discovered.
- **Source Routing + Wormhole**: The source determines a path (potentially from an "ant" or "bee" exploration), then sends data flit by flit to the next node.
- **Destination-Based Routing + Wormhole**: Each node consults a local table (built by "snake" or "spider" sweeps) to direct the head flit.

Practical Considerations

1. **Overhead vs. Benefit**: While biologically inspired discovery can adapt well to changing conditions, each approach introduces control packets or additional logic.
2. **Hybrid Approaches**: Some systems run an "ant-like" algorithm for local path optimization but periodically launch "snakes" for global verification, then use wormhole routing for actual data transfer.
3. **Hardware Feasibility**: On-chip routers with 4, 6, or 8 ports must have carefully sized buffers. Wormhole routing can save SRAM but demands robust flow control to avoid deadlocks.
4. **Complex Resource Management**: Virtual channels, flit-level flow control, and dynamic reconfiguration add to design complexity,

though they allow for higher performance and flexibility.

Conclusion

As chiplet-based systems continue to evolve, combining unconventional, biologically inspired methods of topology discovery (ants, bees, snakes, spiders) with specialized data transfer techniques like wormhole routing ("worms") opens up promising new frontiers. This fusion can lead to adaptive, resilient, and efficient on-chip communication—crucial in an era of many-chiplet systems requiring rapid reconfiguration and minimal latency.

Each of these paradigms underscores a key principle: localized, iterative learning and distributed adaptation can collectively produce robust global outcomes. Layered on top of a suitable forwarding mechanism—whether store-and-forward, cut-through, or wormhole—these biologically inspired methods help chiplet interconnects gracefully handle complexity, partial failures, and reconfigurations, ultimately paving the way for faster and more reliable on-chip networks.

PART TWO: Connection with Bufferless and Deflection Routing

Below is an overview of how biologically inspired "scouting" or "discovery" mechanisms connect with key ideas in bufferless (hot-potato) routing and deflection routing, especially in mesh-like topologies where routers have ports arranged along the cardinal (N, E, S, W) and intercardinal (NE, SE, SW, NW) directions.

1. Biologically Inspired "Scouting" and "Routing"

Local decisions and emergent global organization:

- **Scouting/Discovery Phase**: Biologically inspired methods (e.g., ant-colony-inspired or pheromone-based algorithms) often employ "scout" packets or "explorer" agents that roam the network. These scouts collect local congestion or path-quality information and deposit some form of "trail" (akin to pheromones).
- **Emergent Routing Table Updates**: Each router or switch updates local routing information (sometimes called a local "pheromone table"). Over time, paths that prove consistently "good" get reinforced; less efficient paths fade.

Relevance to On-Chip or 2D Mesh Topologies:

- **Local Compass Directions**: In a regular mesh (e.g., 2D grid) or torus, each router has up to 4 (N, E, S, W) or 8 ports (adding NW, NE, SW, SE). A biologically inspired algorithm can treat each output port as a possible "direction of travel."

- **Natural Fit for Scouting**: The local directional structure matches how "ants" or "foraging agents" might look around in each direction, choosing a route based on local pheromone levels (akin to local congestion or link utilization).

2. Bufferless (Hot-Potato) Routing

Basics of Bufferless Routing:

- **No Packet Buffers (or Very Limited Buffers)**: Every router either immediately forwards or deflects each incoming packet.
- **Hot-Potato / Deflection Character**: If the preferred output port is unavailable, the packet is sent out another (less ideal) port.

Connection with Biologically Inspired Approaches:

- **Continuous Movement**: Biologically inspired scouts are designed to wander; in a bufferless system, "wandering" (via deflections) is also central.
- **Adaptive Reinforcement Over Time**: Pheromone or congestion metrics guide most packets down better ports, even if some must deflect.

3. Deflection Routing

How It Works:

- **Forced Misrouting / Deflection**: If the best output port is busy, the packet takes an alternative route.
- **Common in Low- or No-Buffer Architectures**: Used when buffering is not an option.

Compass Ports Integration:

- **Local Prioritization**: A packet heading NE may prefer N or E, deflect to NE, or in worst cases, NW/SE.
- **Biologically Inspired Ranking**: Use pheromone levels to rank output directions and pick the best available port.

4. Example Flow in an 8-Port Router

1. Receive a packet from the south port.
2. Look up destination (e.g., north-east direction).
3. Check local pheromone table: NE is preferred.
4. If NE port is free, forward the packet.
5. If NE is busy, try N or E or fallback ports.
6. If all preferred ports are busy, deflect to any open port (e.g., SW).
7. Update pheromone levels based on eventual delivery success.

5. Literature and Further Reading

- **Hot-Potato Routing / Deflection Routing:**

  - Baran, P. (1962). *On Distributed Communications Networks*. IEEE Transactions on Communications.
  - Dally, W., & Towles, B. (2004). *Principles and Practices of Interconnection Networks*.
- **Biologically Inspired / Ant-Based Routing:**
  - Di Caro, G. A., & Dorigo, M. (1997). *AntNet: Distributed stigmergetic control for communications networks*. JAIR.
  - Schoonderwoerd, R., et al. (1996). *Ant-based load balancing in telecommunications networks*. Adaptive Behavior.
- **NoC with Deflection / Bufferless Routing:**
  - Moraes, F. et al. (2004). *A Low Area Overhead Packet-switched Network on Chip*. SBCCI.
  - Fallin, C., et al. (2012). *CHIPPER: A Low-Complexity Bufferless Deflection Router*. HPCA.

6. Concluding Remarks

- **Shared Tenets**: Both biologically inspired scouting and deflection routing use localized decision making to produce emergent behavior.
- **Complementary Mechanics**: Pheromone feedback integrates naturally with bufferless routing decisions.
- **Directional Routing**: Cardinal and intercardinal ports align well with the 2D mesh and natural directional behavior in biological metaphors.

Overall, if you combine a scouting mechanism (to adaptively find good routes) with deflection routing (to handle buffer constraints or high contention), you get a dynamic, emergent routing system in which packets flow continuously and local updates shape global traffic patterns in a self-organizing fashion.

4.8 From Hesham

I am more familiar with Ant Colony Optimization Algorithms (ACO) which are used in many applications e.g.

1. Energy and thermal aware mapping for mesh-based NoC architectures using multi-objective ant colony algorithm
2. Ant Colony Optimization-Based Adaptive Network-on-Chip Routing Framework Using Network Information Region

4.9 Design Principles

Came from KAO-DAE-Specifications/Sections/DesignPrinciples.tex

The Daedaelus Substructure (DS) is made of Cells (physical nodes, complete with processor, memory, and FPGA-Enabled SmartNICs),

and physical Links which run the DSPS Protocol, and are based on the following principles:

Every cell has a constrained 7 ± 2 *ports per cell* [1], which balances the need for enough path diversity between any pair of cells to make partitions extremely rare while bounding the complexity of the algorithms for routing, healing and recovery. However, the most significant advantage of a constrained number of ports (valency) is in the building of long-lived Treez and their extended paths (dendrites) which can be used for self-organization[2].

[1] Typical servers today have from 6-8 ports; with an additional 2 ports are used for connecting to Top of Rack (ToR) switches

[2] For example, for migrating data and computations automatically, based on local only information, to create chains or gradients of, for example, data temperature, or the automatic layout of graph computations, or evolving topologies for deep learning applications.

Local information only Each cell has information only about itself and the cells its ports are connected to. While this limitation rules out certain global optimizations, it enhances flexibility in that only neighbors of a cell need deal with changes made to that cell. For example, adding a link only affects the two cells it connects. Local Observer View (LOV).

Failure detection with DSPS provides Temporal Intimacy (The `TIKTYK TIK`, The I Know That You Know That I Know property). `TIKTYKTIK` simplifies the enforcement of desired properties, such as in-order message delivery (even when a failure forces subsequent messages to take a different route to their destination), and Atomic Information Transfer (AIT).

Event driven The network fabric is *event driven*, *i.e.*, there are no network-specific timeouts. Self-sustaining events are created in the `links` using the ENTL mechanism. This is in contrast to conventional computing, where all events are driven by processes running on one or more cores of a processor.

Every cell builds a spanning tree with itself as root A spanning tree provides certain guarantees. Each root can reach all other cells in the fabric with a *tree cast*, and each cell can reach the root with a *root cast*. In addition, it is easy to enforce in-order delivery of messages between any pair of cells, simplifying many distributed computing algorithms.

Each link keeps state needed to recover from routing around its failure Link state is maintained by the two ports it connects to. This information is used to reestablish the ordering guarantees when routing around the link if it fails, to maintain complimentary AIT state on each end of the link, and to silently maintain "presence" for the two parties.

Each cell keeps state to aid in routing around failures The spanning trees (black trees) are built on the `groundfabric` (the physical connectivity), and all cell trees together represent the `groundplane` which is a

connected undirected logical graph that matches the underlying `groundfabric` exactly. When building the spanning trees, each cell records the state of each port for each tree in the fabric. For each tree one port points to a parent, others point to zero or more children, and the rest are not part of the tree. These ports have been *pruned*. We call this state a *TRAPH*, for Tree-Graph. When the link connected to the port pointing to a parent fails, the cell uses the TRAPH data to find an alternate port to reach the new parent.

`groundplane` is the lowest sub-level of the FPGA Substructure : the physical layer. All TRAPHs (selected graph covers) are built as subsets of the `groundplane`. Each TRAPH has a controlling tree (the owner), based on the cell which created and named the TRAPH. This owner cell may keep itself as the sole entity which controls the TRAPH, but more typically, it will select one or more other cells to provide failover should it fail. The failover protocols (Tree Paxos) are described in a separate section of this document.

A recursively stacked set of logical TRAPHs may be built on the `groundplane` to provide the logical segregation layers for secure provisioning and confinement for management entities, such jurisdictions, tenants and sub-tenants. These will often be referred to as the *grey* layers, because they are in the substructure, and are hidden from the hypercontainers above (which run on the other side of the PCIe bus in the hose processor).

Virtual (Colored) TRAPHs may be stacked on top of any of the logical gray layer TRAPHs. Virtual TRAPHs are available under API control for the operating systems and applications above.

4.9.1 Relative Addressing

A major distinction from conventional data centers is that ECNF addressing is based on cell-to-cell (C2C) direct connections. Every communication from an agent on a cell is to one or more ports. The message is sent to the port on the other side of the link, which can forward the message on one of that cell's ports just as switches do.

This addressing scheme has a number of advantages. For example, each cell needs to maintain limited routing information. Should the path to some target agent change, perhaps because it migrated to another cell or due to a failover, only a limited number of cells need to update their routing information. Additionally, an attacker who penetrates a cell can only address its direct neighbors, limiting any potential damage

4.9.2 Port Labelling

Some failover algorithms use path information in the form of a sequence of port identifiers between some cell and the root. Each cell is free to assign arbitrary labels to its ports as long as each of its ports is assigned a label unique to the cell. However, debugging and network management will be easier if a consistent convention is used, as shown in Figure **??** for a cell with 12 ports.

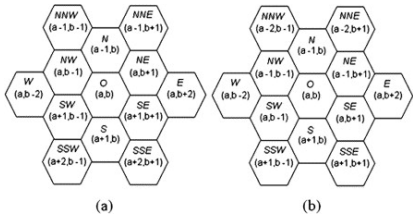


Figure 4.1: Port Labeling convention: 6–12 ports

4.9.3 Hop count

Knowing the number of hops from a given cell to the root can help in selecting shorter failover paths. However, there is a cost. When the path from a cell to the root changes, other cells in the TRAPH may need to be notified to update their hop counts.

4.9.4 Path information

When a link breaks, LW must find a new path to the root of every tree that uses the now broken link. Hop count helps guide the selection of which port to try next, but the new path must be tested all the way to the root. Keeping path information in the form of the each cell's port identifier on the path from the root can reduce that cost. As with hop count, other cells in the TRAPH must be notified to update their path information following a failover.

While messages carrying this information grow with the length of the path, the size is manageable. If we use four bits to label each port, and we use 1K of a 1,500-byte packet for path data, then we can manage 2,000 element paths with a single packet. With balanced trees in a datacenter with N cells, the longest path will be $\log_8 N = \frac{1}{3}\log_2 N$, which is far less than 2,000 for any datacenter envisioned today. Even with unbalanced trees grown stochastically, the longest path is likely to be $O(\sqrt{N})$. We can choose to truncate the path information at the cost of slightly somewhat more attempts before finding a new path.

Introducing Kleinberg links makes the problem tractable even with 64-byte packets, since the average hop count is around 5. Of course, some paths will be longer, but taking 10 bytes of the packet for path information allows us to handle up to 20 hops. We can truncate the path information of the few paths that are longer.

4.9.5 Packet-level Time Reversal

When a recipient of a packet associated with an AIT token cannot pass the packet to the next higher level in the stack, it implements local *time reversal*. The result is that the cells on the two sides of the link end up in the same state they were in before the packet in question was sent. This procedure can be extended to any number of intervening nodes.

This section is premised on the fact that we want to do this kind of time reversal when a link breaks, but I'm not sure why. It would seem that LW only needs to know if it should send the last packet sent before the failure to J, the join node responsible for reestablishing message ordering. No time reversal needed.

4.10 Introduction

Came from ./AE-Specification-ETH/standalone/Graph-Algorithms.tex

This document proposes a Layer 2 Ethernet routing protocol called *Scouting at Layer 2*, inspired by recent advances in edge coloring algorithms from graph theory. It aims to enable deterministic, loop-free path discovery and forwarding solely at the MAC layer, bypassing the need for Layer 3 mechanisms such as IP routing.

Ethernet is traditionally a broadcast-based Layer 2 protocol, relying on IP-based Layer 3 protocols for routing. However, modern data centers and specialized networks demand low-latency, deterministic, and topology-aware communication mechanisms without the full overhead of the IP stack. To this end, we introduce *Scouting at Layer 2*, a distributed routing protocol that operates entirely within the Data Link Layer.

The design is motivated by recent work on edge coloring of graphs in near-linear time Bernstein et al. (2025), which provides a scalable and collision-free scheme for link differentiation.

4.11 Design Goals

- **Layer 2 Only**: Operates exclusively using MAC addresses.
- **No Broadcast Storms**: Avoids STP/RSTP flooding and enables deterministic paths.
- **Loop-Free Forwarding**: Guarantees no cycles using path identifiers.
- **Dynamic Topology Support**: Accommodates changes in network structure.
- **Low Computational Overhead**: Efficient enough to run in SmartNICs or ASICs.

4.12 Graph-Theoretic Inspiration

In the protocol, each Ethernet node and its direct connections are modeled as a graph $G = (V, E)$, where:

- Vertices V are MAC-layer devices (bridges, switches, NICs).
- Edges E represent Ethernet links.
- Each edge is assigned a **color**, i.e., a unique local forwarding tag, such that no two edges incident to the same vertex share the same color.

Using the result of Li et al. Bernstein et al. (2025), we can perform this coloring with at most $\Delta + 1$ colors in $O(m \log \Delta)$ time, where $m = |E|$ and Δ is the maximum degree.

4.13 Protocol Description

4.13.1 Initialization Phase

Each node performs neighbor discovery and assigns temporary colors (tags) to its outgoing links, ensuring local uniqueness. Nodes then gossip their tag assignments to neighbors until a global stable coloring is reached.

4.13.2 Path Discovery Phase

To reach a given MAC address, a node constructs a sequence of tags (path identifiers) describing a color-consistent path through the network. These path sequences are constructed in a Dijkstra-like traversal with color-awareness to avoid collisions.

4.13.3 Frame Forwarding

Frames are modified to include a *Path Identifier Sequence (PIS)* field, which encodes the list of edge colors (tags) to follow. As the frame traverses the network:

1. The switch reads the next tag in the PIS.
2. It matches this tag to an outbound port.
3. It decrements the PIS and forwards the frame.

This process continues until the PIS is empty, and the destination MAC is reached.

4.14 Frame Format

Field	Length (Bytes)	Notes
Destination MAC	6	Standard MAC
Source MAC	6	Standard MAC
Type/PIS Identifier	2	Ethertype or custom
Path Identifier Sequence	Variable	Encoded tag list
Payload	Variable	As usual
FCS	4	Standard CRC

4.15 Advantages

- No need for Layer 3 routing tables.
- Supports programmable switching (e.g., in SmartNICs or eBPF).
- Scales to large networks with sparse connectivity.
- Deterministic pathing avoids congestion and loops.

4.16 Challenges

- Path sequence length is limited by MTU.
- Requires coordination to avoid inconsistent tag assignment.
- Topology changes require propagation of new path info.

4.17 Applications

- HPC clusters with low-latency mesh topologies.
- Edge compute zones with fixed link-layer infrastructure.
- Datacenter overlays where IP is inefficient or unavailable.

4.18 Conclusion

By leveraging edge-coloring strategies for deterministic path discovery and forwarding, Scouting at Layer 2 offers a novel approach to MAC-layer routing. Its foundation in recent algorithmic advances such as those by Li et al. Bernstein et al. (2025) provides a robust and efficient scheme, particularly suited to programmable and high-performance environments.

5. Reversible Transactions

5.1 Mathematical Foundations

Adapted from sections of ./AE-Specifications/chapters/@GPT-reversibility.tex

In low-latency, high-throughput Layer 2 environments (e.g., Ethernet links), it's useful to model transactions as mathematical operations that can be precisely undone. This enables rollback, audit, and error recovery without heavyweight protocols.

1. **Model data as vectors.**
   Each Ethernet frame is viewed as a vector in $\mathbf{GF}(2)^n$, treating bits not as opaque payload but as elements in a vector space over a finite field.
2. **Transactions as invertible operations.**
   The sender and receiver maintain a shared state $S \in \mathbf{GF}(2)^n$. A transaction is an invertible linear transformation T applied to that state: $S' = T(S)$. Because T is invertible, the original state can always be recovered via T^{-1}.
3. **Reversibility via state updates.**
   To reverse a transaction, one sends a message (or derivable signal) allowing the application of T^{-1}. This guarantees deterministic rollback.

We consider a chain of $N+1$ nodes labeled $A_0 \to A_1 \to \cdots \to A_N$, where each node A_i maintains a local state vector $S_i \in \mathbf{GF}(2)^n$, typically initialized to the all-zero vector 0^n or some other agreed-upon state. Each link $(A_i \to A_{i+1})$ between adjacent nodes is associated with an invertible linear transformation $T_{i,i+1}$, which governs how state updates propagate along the chain.

5.1.1 Forward Execution

To execute a transaction spanning all links:

1. At each hop i, node A_i applies $T_{i,i+1}$ to its state S_i and transmits the transformation to A_{i+1}.
2. Node A_{i+1} applies the same $T_{i,i+1}$ to its own state S_{i+1}, maintaining link-local consistency.

The result is a chained sequence of transformations:

$$S'_i = T_{i,i+1} \cdot S_i \quad \text{for } i = 0, 1, \ldots, N-1, \quad S'_N = T_{N-1,N} \cdot S_N.$$

5.1.2 Rollback (Reverse Direction)

Reversibility is achieved by applying the inverse transformations in reverse order:

1. Node A_N applies $T^{-1}_{N-1,N}$ to revert S'_N to S_N.

2. It signals node A_{N-1}, which applies $T^{-1}_{N-1,N}$ and then $T^{-1}_{N-2,N-1}$, and so on.
3. This continues up the chain until A_0 applies $T^{-1}_{0,1}$, restoring the original S_0.

5.1.3 Example: XOR-Based Masks

If each $T_{i,i+1}$ is a simple XOR with mask $\Delta_{i,i+1}$, then:

$$S_i \mapsto S_i \oplus \Delta_{i,i+1}, \quad S_{i+1} \mapsto S_{i+1} \oplus \Delta_{i,i+1}.$$

Reversing just involves reapplying the same mask due to $\Delta \oplus \Delta = 0$.

5.1.4 Notes on Synchronization

- **Acknowledgments:** Each node should confirm that the next node has applied its transformation before committing its own.
- **Composite View:** The full transaction across N links is a composition:
$$T_{\text{total}} = T_{N-1,N} \circ T_{N-2,N-1} \circ \cdots \circ T_{0,1}.$$
- **Error Handling:** Any failure in transmission or transformation must be detected early, as desynchronization across nodes can compound. Redundant encodings, checksums, or commit/abort protocols may be used.

5.2 Atomic Transactions on Æ-Link

5.2.1 One-Phase Commit

5.2.2 Two-Phase Commit

5.2.3 Four-Phase Commit

5.3 Flow Control and Backpressure

5.4 Transactions on Trees

5.5 Reversible Transactions over Ethernet Links

GPT-Compressed adaptation of ./AE-Specifications-ETH/chapters/@Grok-reversibility-original.tex

Atomic Ethernet supports deterministic reversibility at the transaction level. This enables operations to be undone or rolled back by design—without ambiguity or loss—by embedding invertible transformations into the transmission semantics. Here, we outline a formal foundation for such reversibility using linear algebraic constructs, applied to unidirectional flows over a point-to-point Ethernet link.

5.5.1 Framing Transactions as Linear Operators

Let each transmitted message $m \in \mathbb{F}_2^n$ be a vector in a binary vector space. The sender applies a reversible transformation $T \in GL(n, \mathbb{F}_2)$—an invertible matrix over the Galois field $\mathbb{F}_2$—to produce the transmitted payload:

$$m' = T \cdot m$$

At the receiver, the inverse transform T^{-1} restores the original message:

$$m = T^{-1} \cdot m'$$

This formulation ensures that the transformation is loseless and decodable, and that every transaction can be reversed precisely, assuming both ends share T and agree on a consistent ordering.

5.5.2 Design Implications

Reversibility has several architectural consequences:

- **State Preservation:** Nodes maintain minimal internal state beyond the invertible transformation, supporting rollback without checkpointing.
- **Deterministic Rollback:** If a fault or cancellation occurs, the receiver may return m' along with T^{-1} or an encoded undo message, allowing the sender to revert application state.
- **Causal Provenance:** Every transformation T acts as a causal marker for the transaction's origin and structure. This ensures full verifiability of message lineage and intent.

5.5.3 Failure Recovery and Time Symmetry

In conventional systems, rollback is an afterthought—an external protocol layered above an irreversible transmission substrate. By contrast,

Atomic Ethernet supports *built-in reversibility*, which permits time-symmetric protocols where forward progress and backtracking are mirror images.

A link failure mid-transaction results in an incomplete vector transformation. Since the transformation is linear and invertible, partial data receipt can trigger a retry or reversion without ambiguity. The system may encode a rollback transaction as $-T \cdot m$ or transmit a companion transaction to cancel the original.

5.5.4 Physical and Logical Layer Interface

Reversible transactions reside above the PHY layer, but the encoding scheme must be amenable to in-line streaming and backpressure. Slices are transmitted in fixed-size, entropy-bounded chunks, each marked with the transformation context. Intermediate nodes (in a multi-hop path) may propagate or transform T according to routing decisions, but the final receiver must be able to apply a composite inverse.

[MISSING FIGURE “linear-reversbile”]

5.5.5 Applications and Extensions

This mechanism enables new classes of semantics-aware networking:

- **Reversible Compute Fabric:** Transactions may represent computation steps. Reversal allows rollback of mispredictions or branch divergence in distributed computation.
- **Lossless Failure Handling:** Rather than assuming losses, the protocol treats all disruptions as reversible events, ensuring that no state is committed without an acknowledgment or its inverse.
- **Formal Auditing:** Because each transmission is encoded via known operators, a complete proof-of-history can be generated for compliance or replay.

5.5.6 Mathematical Construction

Let the application payload $d \in \mathbb{F}_2^n$ represent a fixed-size bit vector, e.g., 512 bits for a standard Ethernet frame. We model d as a column vector in a binary vector space.

To encode the transaction, the sender selects an invertible matrix $T \in GL(n, \mathbb{F}_2)$, producing the transmitted record:

$$r = T \cdot d$$

where:

- T is an $n \times n$ matrix, generated such that $\det(T) = 1$, i.e., T is invertible.
- The matrix T may be predefined, negotiated, or derived from a seed agreed by both parties.
- The receiver recovers d by applying the inverse: $d = T^{-1} \cdot r$

Example: Suppose $d = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ and the transformation matrix is

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Then the transmitted vector becomes:

$$r = T \cdot d = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

This reversible encoding guarantees that no information is lost in transmission, and rollback is always feasible.

5.5.7 Encoding Strategies

The matrix T can be chosen to reflect transaction metadata:

- **Semantic Operators:** Specific rows or structures in T may denote specific transaction types (e.g., write, abort, checkpoint).
- **Forward-Only Compression:** Even when reversal is not expected, the matrix framework enables low-entropy encoding and structured inference.
- **Replay Protection:** Nonces or keys can be embedded into T to thwart reapplication of stale transactions.

5.5.8 Operational Summary

Encoding Sender maps data d to transmitted record $r = T \cdot d$

Transmission Frame r is sent with metadata identifying or referencing T

Reception Receiver decodes via T^{-1} to recover d

Rollback If reversal is needed, transmit inverse operation using $T^{-1} \cdot r$ or explicitly send a rollback instruction encoded via a known T_{undo}

This mechanism ensures that every transaction in the link pipeline has a symmetric undo path, forming the basis for reversible computing and auditable networking semantics.

5.6 From Clos to Graph: A Shift in Systems Thinking

Conventional datacenter networks built on Clos topologies and Kubernetes orchestration suffer from layered human dependencies: from manual switch configuration to YAML sprawl, from static policy declarations to failure-prone operational recovery. These fragilities compound superlinearly at scale.

Æthernet reimagines this by introducing an alternative: a Direct-Connected 8-Valency IPU mesh, running a decentralized Graph Virtual Machine (GVM). This system inverts operational complexity—delegating routing, scheduling, and security to topology-aware algorithms executed on a uniform graph structure. The goal is zero-trust-by-default, self-partitioning, and human-optional orchestration.

5.7 The Graph Virtual Machine

At its core, the GVM exposes programmable primitives for graph manipulation:

- `traverse(type, source, target)` – initiate message routing via BFS, DFS
- `partition(method, subgraphs)` – divide the network using k-means, MST, or spectral cuts
- `optimize(metric, scope)` – apply shortest path, max flow, or latency tuning globally or locally
- `deploy(tenant, subgraph, policy)` – install workloads subject to affinity, proximity, or isolation constraints

Every instruction is compiled into local actions at each node. No global controller is required.

5.8 Autonomy at 10,000 Nodes

Unlike Kubernetes, which becomes brittle beyond 1,000 nodes due to centralized control planes, the GVM scales linearly. Each IPU communicates only with 8 neighbors and executes a local GVM. Graph operations (e.g., BFS) run in $\mathcal{O}(\log n)$ time.

Maintenance, routing, and failure handling are executed as streaming graph updates. No YAML, no kubectl, no overlays.

5.9 Security via Confinement and Covers

GVM supports autonomous security through:

- **Graph Covers:** Define disjoint regions of the graph to isolate tenants.
- **Stacked Trees:** Construct independent spanning trees for redundant broadcast or failover paths.
- **Dynamic Partitions:** Tenants exhibiting anomalous behavior are moved to smaller or isolated subgraphs.

Attack surfaces are localized to 32 nodes at most. No blast radius extends across racks.

5.10 AI-Augmented Programming

AI assistants integrated with GVM support:

Policy Translation Turn human intent into graph ops (e.g., "keep tenant A near B but far from C").

Optimization Hints Suggest `optimize(flow)` or `partition(k-means)` based on real-time metrics.

Security Monitoring Isolate compromised nodes via AI-derived instruction chains.

Graph Debugging Visualize topologies and bottlenecks, offering suggestions for rerouting or partitioning.

The AI becomes the "network engineer," shifting the cognitive load away from humans.

5.11 Resilience via Topology

The 8-valent IPU mesh has no switches—only direct node-to-node links. This architecture achieves:

- **High Dynamic Laplacian Resilience (DLR):** With 8 neighbors, nodes tolerate up to 7 link failures before isolation.
- **No Leaf Bottlenecks:** Unlike Clos networks, where a failed top-of-rack switch can isolate 32 nodes, this mesh has no such point of failure.
- **Self-Healing:** Graph operations reconfigure routing paths in microseconds.

5.12 Mars-Scale Simplicity

In constrained environments—e.g., Martian colonies—human labor is scarce and risk is intolerable. GVM's autonomy makes it ideal:

- **Resilient:** Survives link loss, power failures, or electromagnetic damage
- **Minimal Ops:** Colonists specify goals; GVM and AI enact them
- **Self-Scaling:** From 100 to 10,000 nodes with no added complexity
[MISSING FIGURE mars-network.pdf]

5.13 Why This Replaces Kubernetes

Metric	Kubernetes/Clos	GVM/IPU
Configuration	Manual YAML, CNI plugins	Self-partitioning graph ops
Operation	Human-tuned schedulers	Fully autonomous
Security	Declarative + fragile	Dynamic + confined
Latency	$10-30\,\mu$s typical	$\sim 50-100$ns
Failure Domain	Rack-wide (32+ nodes)	1–8 nodes max
Scaling	$n \sim 1,000$ ceiling	$n \geq 10,000$ trivial

5.14 Rethinking Atomicity: Counterfactual Transactions

From ./AE-Specifications-ETH/standalone/Transactions-maybe-dup.tex

This document challenges the Forward-In-Time-Only (FITO) assumptions behind conventional transactions in distributed systems. It argues that atomicity, as currently conceived, is a flawed abstraction and proposes a framework for reversible subtransactions as a more robust alternative.

> *"Transactions begin and they end."*
> —Charlie Johnson, TMF Product News

This simple phrase conceals deep design hazards. Transactions appear to begin with a trigger and end with a commit, but in distributed systems, these bookends obscure severe internal inconsistencies.

At issue are the mechanisms we use to track and guarantee these transactional intervals: timestamps, logs, filesystems, and even our concepts of causality. Each introduces cracks in the facade of atomicity.

5.15 The Forward-In-Time-Only Fallacy

FITO: Forward-In-Time-Only thinking assumes linear causality.

Most distributed systems today adopt what we call **Forward-In-Time-Only (FITO)** thinking. That is:

1. **Open a transaction** with a timestamp.
2. **Apply a sequence** of operations.
3. **Close the transaction** with a commit or rollback.

But this approach breaks down under scrutiny.

Three FITO Hazards

1. **Timestamps are not unique.** Even on a single machine with GHz processors and nanosecond clocks, timestamp collisions occur. OS-level clock management does not guarantee uniqueness.
2. **Timestamps are single points of failure.** Any drift, packet loss, or sync error in NTP/PTP introduces false ordering assumptions.
3. **Simultaneity is an illusion.** Relativity tells us simultaneity is observer-dependent. Building global event orderings on timestamps is unsafe.

5.16 The False Comfort of Atomicity

We often say: "all or nothing." But our stack is built on sand:

- The database relies on the log.
- The log relies on the filesystem.
- The filesystem relies on `fsync`.
- `fsync` relies on storage hardware.

Each of these layers fails to guarantee true atomicity. When one fails, the recovery model becomes: **Smash and Restart**.

5.17 The Myth of Reliable Commit

Protocols like Two-Phase Commit (2PC) attempt to enforce distributed agreement. But they depend on:

- Log synchronization across nodes
- Network reliability
- Time-based coordination

When any assumption breaks, so does safety. Eventually, we replace consistency with survivability—and correctness with heuristics.

5.18 Toward Reversible Thinking

Suppose we reject FITO. Suppose we view the transaction as *reversible*.

> If the forward protocol is correct, we can construct a reverse protocol.

This leads to **reversible subtransactions**: bounded operations that can be undone without global rollback.

Counterfactual Transactions

- A transaction can end, then begin again.
- Logs become braids, not linear sequences.
- Atomicity becomes a constraint, not an assumption.

Inspired by Marletto's counterfactual physics, this model embraces partial reversibility as an engineering practice.

5.19 Closing the Interval—Reopened

Simultaneity is not fundamental. Causality is.

"Closing the interval" with a commit only makes sense if we know the state is stable. In reality, it's a guess based on layers of non-atomic operations.

By rethinking transactions through reversible logic, we can:

- Define precise causal dependencies
- Undo partial effects
- Recover without restart

Reversibility isn't science fiction. It is what rollback always wanted to be.

5.20 Conclusion

The abstraction of atomicity has outlived its usefulness as a guarantee. In modern distributed systems, FITO thinking and timestamp dependency introduce hazards we can no longer ignore.

It is time to engineer **reversible protocols**, built on causal semantics—not illusions of simultaneity. Let us design transactions that don't just commit or roll back, but that can *unwind*.

5.21 Reversible Transactions over a Single Ethernet Link

Adapted from ./AE-Specifications-ETH/@GPT-reversibility.tex

Traditional transaction protocols over Ethernet assume a forward-only time model. Failures are handled through retries, timeouts, or speculative execution. This results in complexity, energy waste, and difficulty in verification. Inspired by physics, we propose a model in which every Ethernet transmission is reversible: a transaction is not committed until its inverse is impossible.

Vector-Based Framing

Rather than treating Ethernet frames as opaque byte sequences, we treat them as elements of a vector space over $\mathbb{F}_2^n$. This allows each transmission to be encoded as a linear transformation, and, crucially, inverted.

- A single frame becomes a vector $v \in \mathbb{F}_2^n$.
- A transaction is a linear operation $T : v \mapsto Tv$.
- Inverse operations T^{-1} can be issued on failure, restoring prior state.

Two-Phase Semantics

Each transaction operates in two stages:

Phase 1: Tentative Forward Transmission

The initiator sends a vector transformation. The responder buffers the transformation but does not apply it semantically until Phase 2.

Phase 2: Acknowledgment or Inversion

If successful, a commit acknowledgment is returned. If failure or timeout, the initiator issues T^{-1}, instructing the responder to reverse the tentative state.

Benefits of Reversibility

- **Verifiability:** All transactions are algebraically closed and auditable.
- **Composability:** Compound transactions can be expressed as chains of linear transformations.
- **Energy Efficiency:** Rollbacks avoid speculative execution and wasted computation.
- **No Silent Failures:** The absence of acknowledgment implies an outstanding transformation requiring cleanup.

Implementation Implications

Æthernet links may cache a small rollback buffer per transaction and implement an algebraic acknowledgment loop. This loop behaves as a "circular snake" — with the head and tail of a transaction circulating on the link until either:

- The transaction is committed (the tail is consumed).
- The transaction is explicitly reversed (the tail eats the head).

This model supports deterministic, lossless computing over unreliable mediums by ensuring all state transitions are recoverable — a local time-symmetric alternative to global distributed consensus.

Mathematical Formulation

Let $\mathbf{v} \in \mathbb{F}_2^n$ represent the contents of an Ethernet frame, modeled as an n-dimensional vector over the binary field. Each transaction is a linear transformation:

$$T : \mathbb{F}_2^n \to \mathbb{F}_2^n, \quad \mathbf{v} \mapsto T\mathbf{v}$$

Here, T is an $n \times n$ binary matrix with full rank (i.e., $\det(()T) \neq 0$ in $\mathbb{F}_2$), ensuring invertibility:

$$T^{-1}T\mathbf{v} = \mathbf{v}$$

Each Ethernet transaction must store the pair $(T, \mathbf{v})$ temporarily, enabling the rollback operation:

$$\mathbf{v}' = T\mathbf{v} \quad \text{and} \quad \text{on failure: } T^{-1}\mathbf{v}' = \mathbf{v}$$

This structure creates a *reversible pipeline*, where only acknowledged transformations become final state.

Transaction Algebra and Composition

Multiple transactions may be composed sequentially:

$$\mathbf{v}_2 = T_2(T_1\mathbf{v}_0) = (T_2T_1)\mathbf{v}_0$$

To rollback to any previous stage, it suffices to apply the inverse of the composed transformation:

$$\mathbf{v}_0 = (T_2T_1)^{-1}\mathbf{v}_2 = T_1^{-1}T_2^{-1}\mathbf{v}_2$$

This algebraic chaining enables deterministic state replay and rollback — suitable for verifying transaction histories without logs.

Interpretation as Homology

A reversible link can be seen as enforcing *cycle closure*. That is, any sequence of transformations $T_1, T_2, \ldots, T_k$ across the link must eventually yield:

$$T_k \ldots T_2T_1 = I \quad \text{(identity)}$$

unless a semantic commit is agreed upon and the cycle is broken. This makes every transaction loop homologically equivalent to zero until commitment. The state machine on each end must therefore enforce that:

$$\sum_{i=1}^{k} T_i \in \ker(\text{Commit})$$

until an acknowledgment collapses the pending cycle into permanent change.

Reversibility as a Link Invariant

Just as error correction preserves information despite physical noise, reversible algebra ensures *semantic reversibility* across potentially unreliable hardware. This enables a protocol invariant:

$$\forall t, \quad \text{State}_{\text{link}}(t) \in \text{Image}(\mathcal{T}^{\pm})$$

where $\mathcal{T}^{\pm}$ is the group of invertible transformations. All observable state transitions fall within this reversible group action.

Acknowledgment as Projection

Acknowledgment is not merely a signal but a *projection operator* $\mathcal{P}$ that collapses the reversible superposition of possible futures into a committed state:

$$\mathcal{P}(T\mathbf{v}) = \mathbf{v}_{\text{committed}}$$

Until this projection is received, the state must be maintained in an entangled (reversible) buffer space — a form of local decoherence management.

Extending Reversibility to Graphs

While the reversible transaction model began with point-to-point (1D) links, it naturally generalizes to multi-node graphs.

Spanning Tree for Determinism To maintain deterministic reversibility in a graph of n nodes, define a spanning tree T_s rooted at a node r. Each transaction along edge (i, j) becomes a matrix T_{ij} with:

$$T_{ij}^{-1} T_{ij} = I$$

The system maintains a log of traversed edges to support backward computation if recovery is triggered.

Multi-path and DAG Tensions Acyclic paths support clean reversibility. In contrast, fully connected graphs (K_n) or cyclic DAGs may complicate rollback semantics unless time-bounded checkpoints are defined. Reversibility can be ensured by constraint:

$$\text{Path}(r \to t) \in \text{Tree}_{\text{epoch}}$$

Where $\text{Tree}_{\text{epoch}}$ rotates to provide redundancy without losing invertibility guarantees.

Valency Constraints and Physical Limits

Most physical Ethernet or SerDes ports limit a node's degree to 8 or fewer. As such, highly connected graphs (K_n for $n > 9$) are infeasible. Instead, the protocol supports:

- **Hypercube Topologies:** Efficient for uniform, fault-tolerant routing under degree constraints.
- **Torus and Mesh Graphs:** Enable deterministic reversibility and spatial locality.
- **Tree-Stacked Subgraphs:** Spanning trees with redundant shadows for failover support.

Dynamic Laplacian Resilience (DLR)

We define a resilience metric for reversible Ethernet graphs under link failures:

$$\mathrm{DLR}(G) = \frac{1}{k} \sum_{i=1}^{k} \lambda_2(G_i)$$

where G_i is the graph after i failures and λ_2 is the second-smallest eigenvalue of the Laplacian matrix (algebraic connectivity). High λ_2 implies robust path redundancy and invertibility.

Snake-Based Circulation Model

In the reversible framing model, the "packet" and its acknowledgment together form a single logical object — a snake of bits that must eventually return to its source or be rolled back. Let:

- $\mathrm{Head}(T\mathbf{v})$ be the transmitted transformation.
- $\mathrm{Tail}(T^{-1})$ be the reversible closure.

Completion occurs when both ends of the transformation exist and collapse:

$$\mathrm{Commit}(T\mathbf{v}) \iff \text{Tail returns within timeout and matches } T^{-1}$$

This model avoids deadlocks and waste. Only transformations that complete the loop can semantically alter state.

Time Symmetry in Protocol Design

All operations are temporally bidirectional unless explicitly committed. This mirrors time symmetry in physics and allows for:

- **Exact Rollbacks:** Even at multi-hop or multi-frame granularity.
- **Auditable Transactions:** Linear algebra ensures closure and reconstructibility.

- **Semantic Isolation:** No side effects are propagated until loop closure.

Implications for Reversible Hardware

The reversibility model has strong implications for hardware design:

- **Frame Buffers:** Temporarily store $(T, \mathbf{v})$ until acknowledgment or rollback.
- **Commit Logic:** Frame commits must be locally verifiable by algebraic identity.
- **Link Loops:** The head and tail of a transaction can circulate on the same physical link, forming a tight loop.

In short links (e.g., chiplets), the "snake" can be fully contained in the round-trip delay of the cable. This permits sub-microsecond full transaction turnaround without global arbitration.

Conclusion: Reversibility as a First-Class Network Property

Reversible computation has long been a topic in theoretical computer science, but with Æthernet, it becomes a foundational property of the link layer. The implications are:

1. **All state transitions are reversible until observed.**
2. **Transactions are modeled as group operations with inverse semantics.**
3. **Deterministic rollback replaces speculative retry.**

This model positions Æthernet as the first Ethernet protocol that enforces reversibility as a *physical layer invariant*, enabling full-stack semantic safety — from API to transceiver.

5.22 Linearizability vs. Hypertransactions

From DAE-Technical/Mulligan Stew.pages

There is no such thing as a real-time order. Wall clocks are illusions; physics knows no absolute now.

5.23 Linearizability and Its Limits

5.23.1 What is Linearizability?

Linearizability is a consistency model for concurrent systems that provides a real-time ordering of operations, making the system's behavior appear as if operations were executed instantaneously at some point between their invocation and response.

Real-Time Order: Operations appear to take effect at a single point between invocation and response.

Consistency and Atomicity: Guarantees transition between consistent states as if operations were sequential.

Indistinguishability: Observers cannot distinguish between the actual execution and a hypothetical, linearized one.

5.23.2 Physical Reality

While this may be intuitive for computer scientists, modern physics rejects any notion of absolute real-time order. Special and General Relativity, as well as Quantum Mechanics, demonstrate that causality is indefinite, and entanglement is non-local.

5.23.3 FITO Thinking

Linearizability is a **Forward-In-Time-Only (FITO)** concept. It presumes monotonic progress, ignoring the need to reverse operations. In reality, reversing steps (like backing out of a one-way street) is essential to ensure availability and resilience.

5.24 Hypertransactions

5.24.1 Reversible Subtransactions in FPGA

Hypertransactions are *reversible subtransactions executed entirely within the FPGA substructure*.

> **Moss defines a transaction as:** *"collections of primitive actions that are indivisible, ensuring consistency even with concurrency or failure."*

Despite decades of effort, the industry has failed to achieve true indivisibility. Instead of enforcing an irreversible timeline, we introduce reversibility into the design.

5.24.2 Nested Transactions and Synchronization

Nested transactions enable nested universes of synchronization and recovery:

- Subtransactions can fail independently without aborting the parent.
- Define "reversibility points" to backtrack from local failures.

We use 'invocation' and 'response' as reversible events to guide ancilla bits in managing forward/backward state machine evolution.

5.24.3 From Locking to Multiway Systems

Moss relies on locking, distributed commit, and deadlock detection. We move beyond this to MultiWay Systems, allowing multiple possible valid outcomes in superposition. Determinism is achieved *on demand*.

5.25 HyperIsolation: A New Atomicity Primitive

5.25.1 Atomicity Without Compromise

DAE overcomes the performance/strictness tradeoff of conventional atomicity:

- Violations become visible and recoverable.
- Cell agents monitor liveliness and reroute stalled links.
- Recovery occurs in 2–4 FPGA cycles (sub-microsecond), appearing as "unbreakable" to host processes.

5.25.2 Reversible Definitions of Atomicity

Where Moss defines atomicity in FITO terms, we extend it:

- Proofs hold in both forward and reverse directions.
- Reverse path is mathematically modeled (e.g., invertible functions in linear algebra).
- Alternative rollback paths to safe states are enabled.

5.26 Consistency Reimagined

5.26.1 Mathematical Consistency

Our consistency primitive is rooted in formal mathematical completeness, compositionality, and simulation. Verified in Mathematica, compiled into Petri nets, and synthesized to Verilog.

5.27 HyperIsolation: A New Isolation Primitive

We introduce an entangled, two-phase commit which:

- Transfers reversible tokens of varying size across FPGAs.
- Overcomes limitations of Serializability and S2PL.
- Prevents host-side blocking via TPI (Transaction Processing Interface).

5.28 Physics and Networking: A Rebuttal to FITO

5.28.1 Real-Time Order is an Illusion

Physics denies the concept of instantaneous global time. All observations are relative, and latency is inevitable.

5.28.2 Consistency and Atomicity Are Observer-Relative

- Transitions appear different to each observer.
- Faults expose inconsistent sub-states that cannot be masked.
- Redundancy and availability metrics do not guarantee determinism.

Entanglement means maintaining consistent-but-complementary states between peers.

5.28.3 Linearization Fails under Fault

Packet drops, reordering, and duplication break coherence between senders and receivers. Indistinguishability is violated. Sequence numbers alone cannot recover lost structure.

5.29 Sequence Numbers as Complex Modulo Counters

- Sequence numbers must be reversible modulo counters.
- Bounded interaction buffers (2, 4, ... N interactions) are provably atomic.
- We use complex-number representations, stored in lookup tables.

Note: Increasing counter width increases failure likelihood and recovery complexity.

6. Architecture

6.1 Back to Back Shannon Channels

From ./AE-Specifications-ETH/Shannon.tex

6.1.1 Two independent Metcalfe Channels (Max flow, no Interaction)

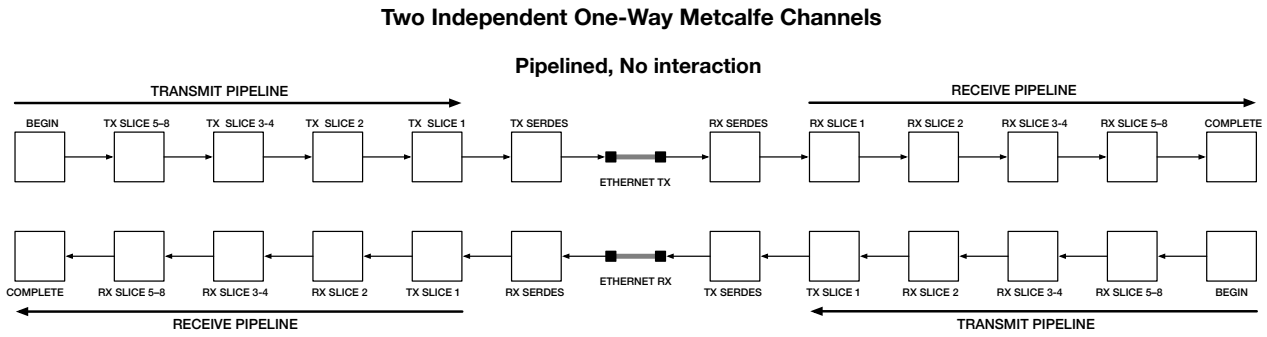


Figure 6.1: Two Independent Metcalfe Channels

6.1.2 Internal (SACK) Feedback on last slice

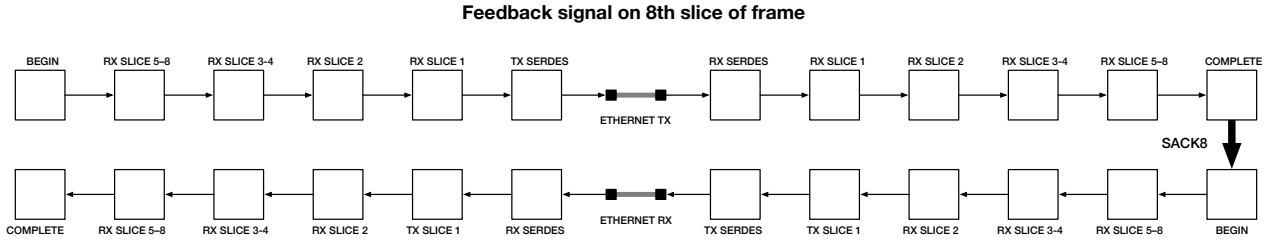


Figure 6.2: Feedback signal on slice 8

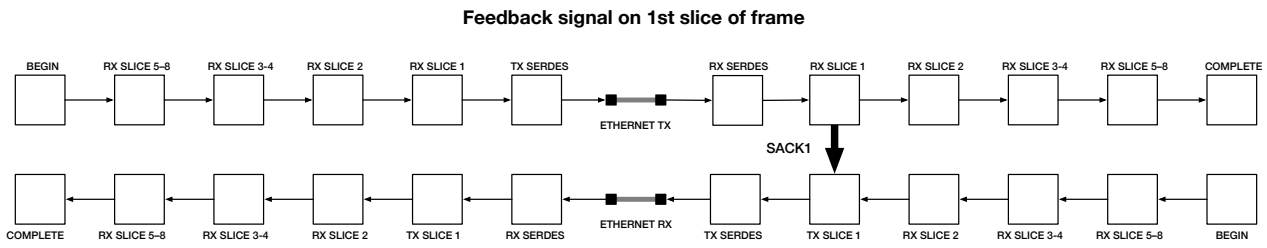


Figure 6.3: Feedback signal on slice 1

6.1.3 Internal (SACK) Feedback on first slice

See SACK description by Sahas

6.2 Architectural Framework: Four Shannon-like Levels

In the proposal for subdividing a 64-byte packet into 8-byte slices, we introduce partial acknowledgments (SACKs) at four (decrementing) boundaries (`11`, `10`, `01`, `00`). Each of these points reveals an incrementally deeper level of the receiver's certainty about the data, the hardware, and the appropriate next step in the protocol. We can interpret this progressive certainty in terms of four conceptual *layers* reminiscent of Shannon's *information* theory, but extended to address knowledge, semantics, and understanding. This layering describes how a receiver (e.g., the SmartNIC) transitions from raw incoming bits to meaningful messages that can be handed off to the host processor.

Back-to-Back (B2B) Shannon Channels are "Perfect Information Feedback" (PIF) as senders see their own transmitted packet returning back from the receiver and thus can detect channel errors. Thus making CRCs Checksums, Parity and FEC unnecessary. Similar to "Perfect Information Feedback" in: Norm Abramson, "Packet switching with satellites," NCC, 1973

6.2.1 Layer 1: Information (Surprisal)

At the first level, information refers to the direct "yes/no" answer to a question of interest: the arrival or non-arrival of bits, which Shannon famously treated as the surprisal of a received symbol. At the SACK `00` boundary, when the receiver detects the first 8-byte slice without error, it learns that the link is alive and that the data matches expectations (i.e., no immediate mismatch). This is pure information because it distinguishes the event "we did receive slice #1 correctly" from "we did not." The mutual information gained here confirms a working cable and a functional SerDes.

At this early stage, the question posed is binary: "Did the hardware see valid bits?" The surprisal is that valid bits were received, as opposed to no signal or corrupted data.

6.2.2 Layer 2: Knowledge (Captured Information)

The second layer, knowledge, arises when the raw bits are stored or captured in a meaningful structure. This could be as simple as a recognized slice stored in buffer memory or a pipeline register. By the time the second slice arrives, the receiver has captured more bits—16 bytes in total—and placed them into NIC-internal registers. It can then perform further checks, such as alignment, partial CRC, or checking for expected header fields. The SACK `01` confirms that the hardware not only saw valid bits but also placed them in the correct buffer location.

At this point, the system has a partial understanding of the data. It knows that the 16 bytes are recognized and safely stored, awaiting deeper logic to interpret them.

6.2.3 Layer 3: Semantics (Meaning)

The third layer, semantics, involves the system deciding what the bits mean in terms of subsequent action. This layer determines which state machine or processing path is relevant for the given data. At the SACK `10` boundary, after 32 bytes have been received, the NIC has gathered enough information to partially decode the data. For example, it might be able to determine which protocol or message type is indicated. The NIC can confirm that buffer slots or ring descriptors are available and that the correct state machine is loaded (e.g., state machine A for small control frames, or state machine B for streaming payloads).

Once the NIC signals SACK `10`, the sender learns that the hardware has found the data coherent enough to continue. The semantics are recognized sufficiently to proceed without hazard. The receiver has now moved from simply knowing the bits are correct (Layers 1 and 2) to understanding how to proceed and which internal resources or state machines to activate.

6.2.4 Layer 4: Understanding (Syntax)

The final layer, understanding, refers to the recognition that the message fits into a finite set of concepts or message types that the NIC accepts. This implies that the message has a correct syntax recognized by the hardware. At the SACK `11` boundary, which occurs when slices 5–8 arrive, the full 64 bytes have been received and match a legitimate frame or message layout. The NIC is now ready to push the message onto the PCIe bus or an internal ring buffer for the host processor.

At this stage, the NIC has a full understanding of the message, knowing exactly how to finalize the packet, classify it, and pass it upstream for higher-level processing. No further layer-2 repairs are needed, and the message is ready for the next step in the protocol.

6.3 Bidirectional Shannon Channels

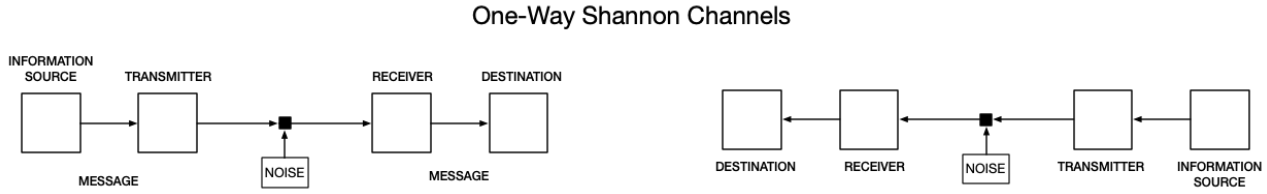


Figure 6.4: Shannon One-Way Channels.

Shannon Channels are normally shown in one *direction* of flow – from Information source to Information Destination. Here we exploit two-way communication (signaling) Back to Back Channels with immediate (slice by slice) feedback. Then the equations tell us something interesting about the *symmetry* of set reconciliation on both sides of the link.

With back-to-back Shannon Channels, with (immediate) slice by slice feedback, we get Perfect Information Transfer (PIT. We can therefore dispense with Checksums, CRC's, FEC or even Parity, because the failure modes these EDC and ECC codes address are already covered by PIT. This has two advantages:

- The elimination of spatial redundancy on the wire makes the packets shorter
- The need to calculate increasingly complex codes reduces computation and energy dissipation on the link

Redundancy is a poor crutch when assumptions about uniform probability distributions are violated (which they almost always are in practice).

6.3.1 Metcalfe Half-Duplex

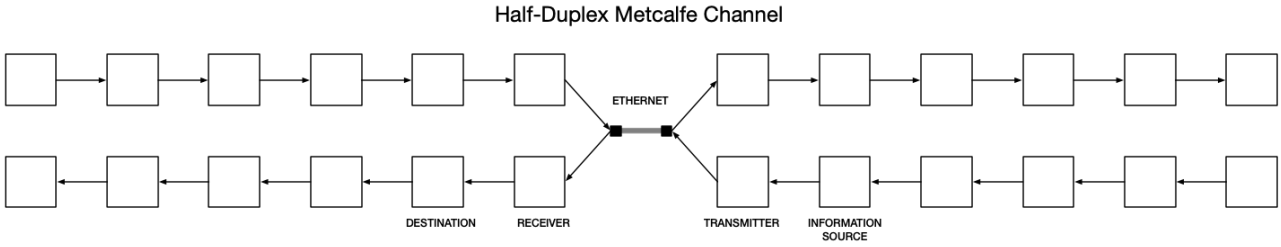


Figure 6.5: The original Metcalfe + Boggs Ethernet was a bus. A long cable where 'stations' were TAPs on the bus. This meant that each station had to both listen, and transmit from teach tap. In this figure we show two *independent* streams of packets going in opposite direction (forwardpropagation and backpropagation) through a single half-duplex link

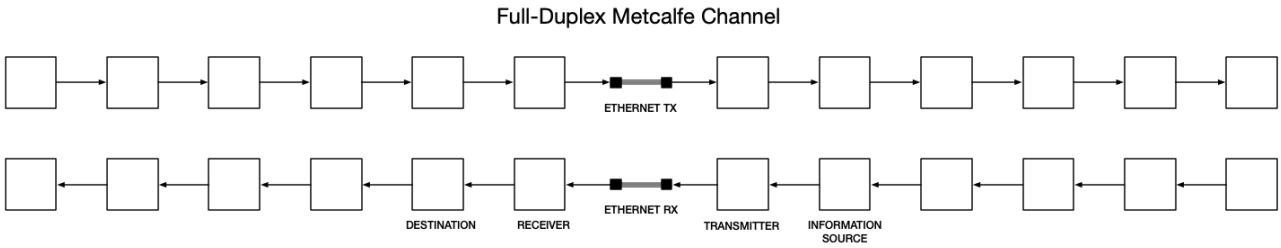


Figure 6.6: Modern Ethernet Links are bidirectional; two sub-channels: one for transmit, one for receive

6.3.2 Full-Duplex Bi-pipelined Shannon-Metcalfe Channel

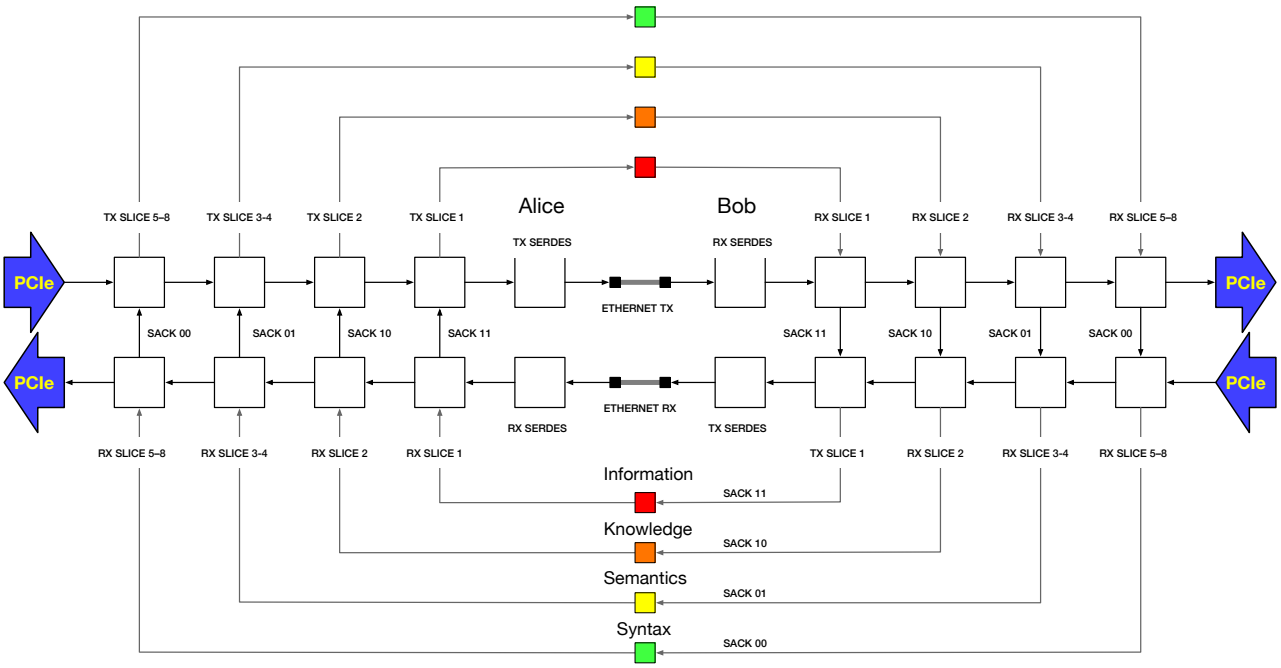


Figure 6.7: Complete model: Bi-pipelined full duplex exchange of Æthernet frames. Complete with internal "Slice ACKnowledges" (SACKs) – sequenced with increasing *common knowledge* depth inside SerDes/FPGA

The figure above is a simple, formally verifiable, mathematical description, from API to bits on the wire (Shannon channel).

6.4 Short-Range ANT (Local) Scouting

./AE-Specifications-ETH/standalone/Scouting.tex

Once a link has been established, they are recorded in the local `knowledge` of the cell, and used as a baseline for future algorithmic and policy decisions. Immediately after establishing a reliable connection, `CELL`s may emit `ANT-SCOUTS` to explore their local environment. These are ANT's, which obey an initial source routing algorithm, but when encountering a failed or disconnected port in another cell, respond with either clockwise, anticlockwse packet forwarding, which keeps the scout local, or *random*, with a hopcount limit, which allows exploration further afield.

See ANT Specification for details

6.5 Long-Range BEE (Global) Scouting

`CELL`s may also emit `BEE-SCOUTS` to explore the extremities of their environment. These are BEE's which obey only one rule: proceed in the same direction as the radial port. BEE's emitted on the *n* port may only go *n*. BEEs emitted on the *se* port may only go *se*. Until they encounter a disconnected port, whereupon they execute a return path algorithm, accumulating information at each `CELL` and returning it to the root.

See BEE specification for details

6.6 ANT Specification: Triangle Packet Clocks in 3×3 Tiles

Packet clocks are initiated by the coordinator, on any of it's active links. The ANT (source routing) algorithm goes out on any port, and are programmed to turn left or right at the first available active port. The convention is turning right makes it go clockwise, and turning left makes it go antilockwise, but this is an artificial distinction. As with real ants, they can get lost, and never find their way back to the nest, and they die (or return to the nest by inverting their source routing paths). This "limited range", is part of the Security mechanism.

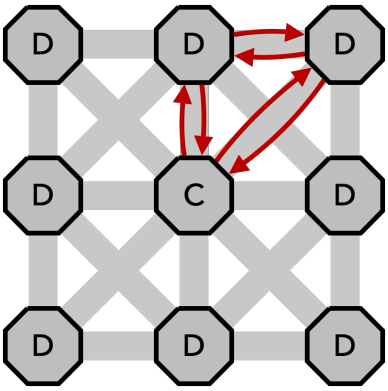


Figure 6.8: Race-Free Triangle Token

6.7 ANT Specification: Race-Free Packet Clocks in 3×3 Tiles

Once the cell has discovered it's local environment, it may establish packet clocks. These are ANTs, which go out with a pre-defined pattern, to return events the cell on a *periodic* basis. Because there is no *background* of time, this system will create events, which are guaranteed to occur without race conditions, but will catastrophically fail if there are an broken links around the circuit.

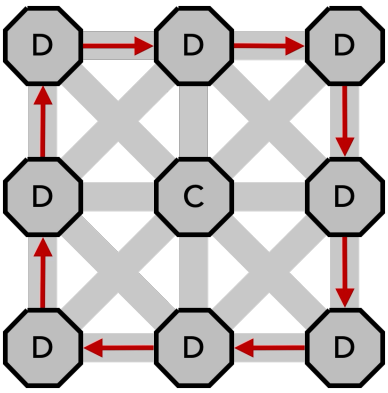


Figure 6.9: Square Race-Free 1-hop Clock

6.7.1 ANT Specification Building a Compass Clock

8 physical ports per cell. Inactive ports may be:

- Failed (out of service)
- Standby, ready to go
- Off, saving energy

Packets clocks may be initiated around the closest (one-hop) cell tiles, next closest (two-hop) tiles, furthest (three-hop) tiles. Atomicity

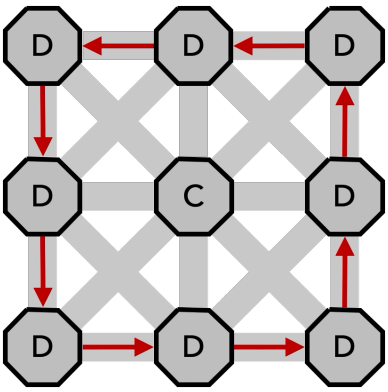


Figure 6.10: Race-Free Anticlockwise

6.7.2 ANT Specification: Counter Circulating Race-Free ANTS

C (Carol, Charlie, Coordinator, Chief) may initiate clockwise, counter-clockwise, and/or both at once. Each is exploring the health of the connectivity local to the center cell. This is what ANT Algorithms (source routed, or random) tokens.

Ports at edge of mesh connected back to same cell on a different port to traverse routing table 2nd time to create virtual cut-through torus.

If the ANT gets blocked, and either runs out of hopcout resource, it does 'reverse path forwarding' back to the C CELL. and reports what it finds. It can either carry all its state in the packet, or (ror BEEs), clean up on its way and erase its footprints in the CELLs it visited.

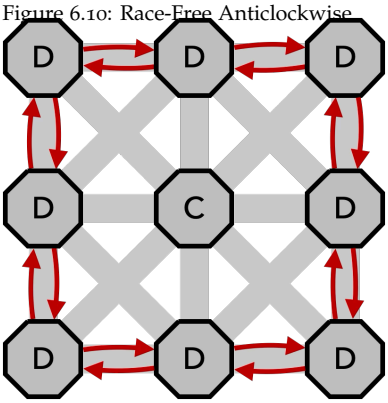


Figure 6.11: 2 Circulating Race-free tokens

6.7.3 7 x 7 Nodes Packet Clock

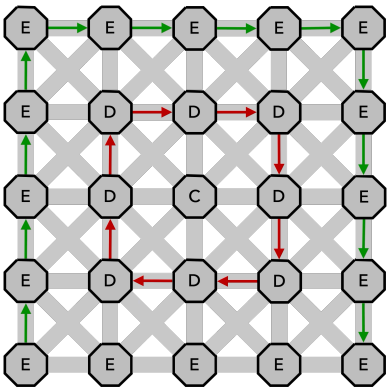


Figure 6.12: Green Packet Rateless Clock

6.8 Beyond Packet Clocks

Packet clocks don't scale (they are not intended to). Instead, they provide circulating logical loops **?**. The local system policy will establish the radius limit for local exploration. Everything beyond that is in the domain of the BEE scouts .

Packet clocks can circulate at any physical hop distance. The one-hop agents are described above. The two figures on the right show an example of an ANT which goes two hops, or three hops, before the ANT turns left or right. This give a CELL the opportunity to explore larger hop distances from the coordinator

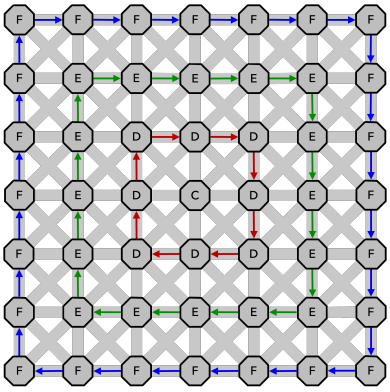


Figure 6.13: 3rd-hop circular packet clocks. Blue Links Complete

6.9 Packet Clocks in Larger Tiles

6.9.1 BEE scouts

BEE Scouts explore the boundaries of their environment. The are emitted by the Coordinator, and travel as far as they can in ONE direction, `{dn,ne,de, se, ds, sw, dw, nw}`, and then *return on the recipro-*

cal path (Compass-Point vector direction) to inform the hive (root) what they discovered, so the root can build it's model of the topology, and Edge resources to perform their function.

6.9.2 N x N Nodes Packet Search Rays (BEEs)

BEEs are radial distance scouting agents. Single packets that go in only one direction, and when they reach the end (extremities of the Cellular interconnect) they execute a reverse path forwarding algorithm, collecting knowledge on their way, delivering this knowledge back to the root, whose agent uses the returned information to build it's model of it's topology and available resources to offer 'services' to the applications.

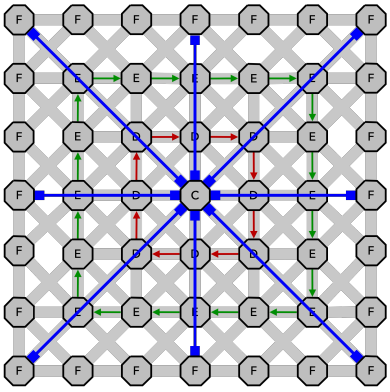


Figure 6.14: BEE Algorithms explore beyond ANT algorithms

These don't have to be square, or rectangular. BEE algorithms work on any arbitrary Topology.

Radial (Ray) source-routed scouts have two parameters (a) which port they go out on, and continue indefinitely until they reach a boundary (or exhaust their hop count resource). And then they return along exactly the same path, accumulating knowledge of the CELLS on their way (e.g. properties of the cell, do they have a CPU, a GPU, an IPU, or QPU?). Most Bees make it back home to the nest (C) but it is also possible for a failure to occur between the outbound BEE and the homebound BEE. In which case the packet try's to make it's way to 'Lost and Found', the control structures identified by the Coordinator to provide GEV notification of failures. Lost and Found is most likely to be discovered by the one or more of the BEEs. Edge nodes (on the corners of the interconnect), will always be able to 'find' Lost and Found (and other external control paths controlled by monitoring or configuration LOGICAL Administrator Agentss) with a 'due north or south' `dn,ds`, 'due-east or west' `de,dw` BEE Scout.

6.10 Local decisions and emergent global organization

- Scouting/Discovery Phase: Biologically inspired methods (e.g., ant-colony-inspired or pheromone-based algorithms) often employ "scout" packets or "explorer" agents that roam the network. These scouts collect local congestion or path-quality information and deposit some form of "trail" (akin to pheromones).
- Emergent Routing Table Updates: Each router or switch updates local routing information (sometimes called a local "pheromone table"). Over time, paths that prove consistently "good" get reinforced; less efficient paths fade. This local, probabilistic approach

can converge on globally efficient routes with no central coordination.

6.10.1 Relevance to On-Chip or 2D Mesh Topologies

- Local Compass Directions: In a regular mesh (e.g., 2D grid) or torus, each router has up to 4 (N, E, S, W) or 8 ports (adding NW, NE, SW, SE). A biologically inspired algorithm can treat each output port as a possible "direction of travel."
- Natural Fit for Scouting: The local directional structure matches how "ants" or "foraging agents" might look around in each direction, choosing a route based on local pheromone levels (akin to local congestion or link utilization).

Thus, the scouting/discovery mechanism is all about gathering local "pathworthiness" data and then directing future traffic toward better routes—exactly how a local compass-based system can easily be integrated.

6.10.2 Bufferless (Hot-Potato) Routing

- No Packet Buffers (or Very Limited Buffers): In a bufferless architecture, every router typically either immediately forwards or deflects each incoming packet. Packets cannot wait in large queues when an output port is congested.
- Hot-Potato / Deflection Character: When the preferred output port is unavailable, the packet is sent out of a different (less ideal) port—"hot-potato" style—rather than being buffered.

6.10.3 Connection with Biologically Inspired Approaches

- Continuous Movement: Biologically inspired scouts are already designed to wander and discover; in a bufferless system, "wandering" (via deflections) is also central. This synergy means a router can apply a heuristic (like a pheromone table) to pick the "best available port" quickly, but if that port is busy, the packet must choose an alternate direction.
- Adaptive Reinforcement Over Time: In a bufferless design, a packet cannot linger while waiting for the optimal output. However, local "pheromone" or "congestion" metrics can still help route the majority of packets down better ports more often. Over time, high-traffic edges might become less appealing, guiding packets to less-congested directions.

6.10.4 Deflection Routing

- Forced Misrouting / Deflection: If the desired or minimal-distance output port cannot be taken (due to contention), the router picks

another output. The packet may travel away from its ultimate destination (a "deflection"), but eventually, it should be re-routed back on track.
- Common in Low- or No-Buffer Architectures: Deflection routing is one way to handle resource contention when buffer space is unavailable.

6.10.5 Tying It Back to the Compass Ports (N, E, S, W, NW, NE, SW, SE)

- Local Prioritization: In an 8-port (or 4-port) router, one can define a strict or heuristic priority among the directions. For example, a packet traveling generally "north-east" might prefer the N or E port if free; if both are busy, it might deflect NE, or in the worst case, deflect NW or SE.
- Biologically Inspired Ranking: The "pheromone" concept can be used to rank the output directions. The highest "pheromone" port is tried first, then so on down the rank. This effectively merges a local heuristic (pheromone) with forced deflection for whichever ports remain free.

In practice, such a scheme allows packets to "scout" and reinforce certain directions while still ensuring that they never have to wait for a blocked port.

6.10.6 Example Flow in an 8-Port Router

1. Receive a Packet coming in from, say, the south port.
2. Look Up Destination (or partial coordinate heading). For instance, the packet is trying to reach a node in the north-east region, so N or E might be favored.
3. Check Local "Pheromone" or Routing Table: Suppose the local pheromone table says port NE is the best guess based on past traffic patterns.
4. If NE Port Is Free: Forward the packet NE.
5. If NE Port Is Busy: Check next best local direction (N, E, or NW/SE fallback).
6. If All Preferred Ports Are Busy: Packet is deflected to any open port (could be even SW in the worst case).
7. Local Table Update: The router sees how that choice ended up affecting the packet (if it eventually left the region quickly or ended up in a congested area). Over time, these experiences feed back into local pheromone levels.

Despite the forced misrouting (deflections), the biologically inspired feedback approach often keeps net throughput healthy and tries to avoid systematic congestion "hot spots."

6.10.7 Connection with the Literature

1. 1. Hot-Potato Routing (Deflection Routing):
   - Baran, P. (1962). On Distributed Communications Networks. IEEE Transactions on Communications. (Early ideas of "hot-potato" and distributed routing).
   - Dally, W., & Towles, B. (2004). Principles and Practices of Interconnection Networks. (Excellent overview of deflection routing in modern network design).
2. 2. Biologically Inspired / Ant-Based Routing:
   - Di Caro, G. A., & Dorigo, M. (1997). AntNet: Distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research.
   - Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. Adaptive Behavior.
3. 3. Network-on-Chip with Deflection/Bufferless Approaches:
   - Moraes, F. et al. (2004). A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping. SBCCI.
   - Fallin, C., et al. (2012). CHIPPER: A Low-Complexity Bufferless Deflection Router. HPCA.

These resources flesh out how bufferless or deflection routing is implemented (especially in on-chip contexts) and how biologically inspired heuristics can be adapted to local, minimal-knowledge scouting decisions.

6.10.8 Concluding Remarks

- Shared Tenets: Both biologically inspired scouting and deflection-based, bufferless routing rest on local decision making. In biologically inspired schemes, scouting packets "discover" or "reinforce" certain paths. In deflection routing, each router makes a quick (local) decision when a preferred port is blocked, forcing packets to keep moving.
- Complementary Mechanics: Because biologically inspired "pheromone" updates naturally reflect congestion and path usage, they integrate well with a bufferless or deflection style—turning forced misroutes into valuable "exploration" signals that feed back into local heuristics.
- Directional Routing: The presence of N, S, E, W (plus diagonals) simply defines how many possible local moves each node (router) can attempt. In 2D meshes or tori, these directions make for a convenient coordinate system that parallels how ants (or other scouts) might sense local gradients or pheromone intensities in each of eight compass directions.

Overall, if we combine a scouting mechanism (to adaptively find neighbors and good routes) with deflection routing (to handle buffer constraints or high contention), we get a dynamic, emergent routing system in which packets flow continuously and local updates shape global traffic patterns in a self-organizing fashion.

All this happens without the need for Source/Destination Addresses, which present severe security problems by exposing the “identity" of nodes making them vulnerable to attack.

6.11 A Blueprint for Æthernet Protocol

From ./section/BSW.tex

6.12 Foundations

We begin with Bartlett, Scantlebury and Wilkinson's (Davies et al. (1967)) Alternating Bit Protocol (ABP).

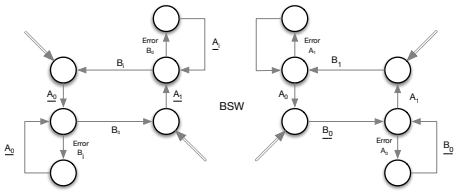


Figure 6.15: Alternating Bit Protocol(ABP)

Alternating messages are implemented as a single snake (a longitudinal set of bits traveling through the wire and FPGA's). Wrapping its way through the SerDes Registers of Alice and Bob on both ends.

First imagine a zero length wire connecting the chiplets for Alice and Bob (they are as adjacent as they can be on a module or motherboard). Two SerDes channels are directly connected.

Where in BSW[1], the edges of the automata are labeled with the 'alternation' bit, Lynch expands the single bit to multiple bits.

[1] A note on reliable full-duplex transmission over half-duplex links

A single bit alternates, but it does not have a direction.

Alternating 2-Bit Protocol (from Lynch):

Two bits at least have a direction in their evolution through their set.

`{00, 01, 11, 10}`

OR

`{11, 10, 01, 00}`

Where the next item establishes the direction of evolution of the set.

We use Ternary logic `-1,0, +1` in our model of the protocol.

o simply means 'equilibrium', which gives us the +1 and -1 complementary states that are expected of Ternary Logic. But there is a mathematical subtlety: we need both two versions of Zero, one approaching from the negative side, and one approaching from the positive side.

This provides a mechanism for *forward-propagation* and , of the state.

6.12.1 Proof of Fallibility

"the alternating validation bit becoming the additionally the *alternation* bit for message transmission in one direction, while the alternation bot for the reverse directions serves additionally as a validation bit" SBW Davies et al. (1967)

Lynch's scheme is constructed from independent simplex procedures

6.13 Alternating Causality

QUESTION: What is the difference between reliable and fallible?

6.14 Framing the Vocabulary

In 1967 *Bartlett, Scantlebury & Wilkinson* (BSW) sketched the *alternating-bit protocol* (ABP): add one history bit to every frame, wait for an ACK that echoes it, and retransmit until the right ACK appears. ABP wraps an *unreliable* medium and presents a service that looks reliable—even *infallible* in steady state.

Nancy Lynch later formalised these ideas with the *I/O-automaton*: a fallible channel is one whose execution may deviate from the specification, subject only to fairness.

Term	Meaning
Unreliable	Frames may be lost.
Fallible	Channel may violate any promise (drop, duplicate, reorder, corrupt).
Reliable	No drops in steady state; recovery still required.
Infallible	No violations ever; no recovery logic needed above the link.

Figure 6.16: Taxonomy of link qualities.

6.15 Where Classical Ethernet Fits

Original 10 Mb/s Ethernet (and most "best-effort" variants since) offers a CRC to *detect* corruption but no link-level retransmission. Frames can be dropped by congestion, policing, or topology loops. Hence raw Ethernet is both *unreliable and fallible*; higher layers—typically TCP—supply ABP-like recovery.

6.16 How InfiniBand Raises the Game

InfiniBand *embeds* ABP into silicon:

- Per-hop *credit flow control* makes buffer overflow almost impossible.
- Link-level CRC plus optional *link retransmission* retries any corrupted frame.
- Reliable Connection and Reliable Datagram queue pairs carry *ACK-/NACK* sequence numbers end-to-end, guaranteeing exactly-once, in-order delivery across multi-switch fabrics.

To software, the fabric appears nearly *infallible*; drops are rare and localized.

6.17 Reliable vs. Infallible, Unreliable vs, Fallible

Table 10.1 highlights the nuance. Priority Flow Control (PFC) can render an Ethernet link *loss-less* in steady state, but deadlock, misconfiguration, or burst congestion can still drop frames. Such a link is "reliable yet fallible." Infiniband's credit + retransmit pipeline, by contrast shifts real-world operation toward "reliable and almost infallible."

6.18 Why Ethernet Still Struggles

1. **Retrofitting**: inserting link retransmission into the IEEE 802 stack breaks long-standing timing and compatibility assumptions.

2. **Congestion domain**: shallow switch queues and ECMP paths leave more surfaces for loss than InfiniBand's strict hop-by-hop credits.
3. **Layering philosophy**: because TCP "already" ensures delivery, many operators accept occasional loss rather than pay silicon cost for hardware recovery.

6.19 Lessons from BSW and Lynch

- *Make reliability local*. ABP attaches one bit; InfiniBand embeds a few more. End-to-end recovery alone expands the failure scope.
- *Fail fast*. InfiniBand retransmits on explicit NACK within microseconds; Ethernet traditionally converts a microsecond drop into a millisecond TCP timeout.
- *Separate reliability from recovery*. Even a reliable link needs a failsafe plan; design that plan explicitly.

6.20 Conclusion

CONCLUSION:

BSW showed that a single alternating bit can tame a capricious wire; Lynch supplied the proof rules. InfiniBand adopted both insights in hardware and delivers a fabric whose normal behavior feels *infallible*. Classic Ethernet remains best-effort—unreliable and fallible—and relies on upper layers for recovery. Bridging that gap means absorbing more of the ABP playbook at the link: credits, link-level retransmission, and tight ACK/NACK loops that shrink recovery from milliseconds to microseconds.

6.21 Forward and Backpropagation Through the Lens of the Two-State Vector Formalism

The *Two-State Vector Formalism* (TSVF), developed by Aharonov and collaborators, describes a quantum system using both a forward-evolving state vector from the past and a backward-evolving vector from the future, forming a complete description of the system between two time boundaries.

This formalism maps intriguingly well onto the two-phase behavior of supervised learning:

- **Forward propagation:** evolving the input forward through the network to predict an output.
- **Backpropagation:** retroactively applying a loss function at the output and propagating error information backward through the network to adjust weights.

Forward propagation plays the role of the forward-evolving quantum state $|\psi(t)\rangle$, and backpropagation corresponds to the backward-evolving dual state $\langle\phi(t)|$. Together, they constrain the learning dynamics at each layer via a two-state viewpoint.

6.22 Forward Propagation as Forward Evolution

In TSVF:

$$|\psi(t)\rangle = U(t, t_0)|\psi(t_0)\rangle,$$

where U is the unitary time evolution operator from an initial preparation at t_0.

In machine learning:

$$a^{(l+1)} = f^{(l)}(W^{(l)}a^{(l)} + b^{(l)}),$$

where the activations $a^{(l)}$ propagate the input forward.

6.23 Backpropagation as Backward Evolution

In TSVF, a post-selected state $\langle\phi(t_1)|$ evolves backward:

$$\langle\phi(t)| = \langle\phi(t_1)|U(t_1, t),$$

complementing the forward-evolving $|\psi(t)\rangle$.

In neural nets:

$$\delta^{(l)} = (W^{(l+1)})^\top \delta^{(l+1)} \circ f'^{(l)}(z^{(l)}),$$

where $\delta^{(l)}$ encodes error at layer l and propagates backward to adjust weights.

6.24 Two-State Update Rule

In TSVF, expectation values take the form:

$$\langle A\rangle_w = \frac{\langle\phi|A|\psi\rangle}{\langle\phi|\psi\rangle},$$

and represent weak values or amplitudes constrained by both past and future states.

In machine learning, the update rule:

$$\Delta W^{(l)} \propto \delta^{(l)}(a^{(l-1)})^\top$$

depends on the forward activations $a^{(l-1)}$ and backward error signal $\delta^{(l)}$. Together, they act like a sandwich operator:

$$\text{Update}^{(l)} \sim \langle\phi^{(l)}|\text{operator}|\psi^{(l)}\rangle.$$

6.25 Time-Symmetric Learning View

Rather than treating backpropagation as a mere computational trick, TSVF offers a time-symmetric interpretation:

- Both the input and the desired output state determine the intermediate learning dynamics.
- Each layer mediates between past input and future supervision, forming a time-bridging node.

6.26 Comparison Table

Concept	Neural Network	TSVF QM
Input	x	$\|\psi(t_0)\rangle$
Process	Forward prop	$U(t, t_0)$
Target	Loss function	$\langle\phi(t_1)\|$
Error	$\delta^{(l)}$	$\langle\phi(t)\|$
Activity	$a^{(l)}$	$\|\psi(t)\rangle$
Update	$\delta^{(l)}(a^{(l-1)})^\top$	$\langle\phi\|A\|\psi\rangle$

Table 6.1: Analogies between supervised learning and TSVF.

6.27 FITO Thinking vs. Time Symmetry

Most machine learning frameworks assume *Forward-In-Time-Only (FITO)* causality: input causes output, and learning proceeds only by adjusting from past to future. TSVF suggests a richer model:

- Supervision from the future constrains the learning of the past.
- This bidirectional model aligns with concepts from goal-driven behavior and active inference.

6.28 Conclusion

The TSVF reframing of forward and backpropagation illuminates the deeper time-symmetric structure underlying learning. Far from being just a computational trick, backpropagation can be seen as a physical dual to forward propagation—both necessary to fully specify a learning system between two boundary conditions.

This is highly relevant to our model of the protocol, where we use reversible state machines to specify forward propagation, with whatever reversible glitches might be needed to handle failures are implemented as reversible steps, and the backpropagation as the credit-based flow control mechanism.

Because they are completely symmetric, packets being sent and packets being unsent are fully managed by the flow control system.

6.29 IP and Patent Implications

The core concept of credit-based flow control is now public domain, but a handful of post-2005 implementation patents remain enforceable through at least 2036. Modern designs should audit those families but can build freely on the expired foundational work.

6.30 Practical Take-Aways

1. **Freedom to Operate.** A straightforward "one credit = one buffer" design can rely on the expired Intel/Compaq and Brocade patents for prior-art cover. Avoid features identical to the still-active patents or license them.
2. **Design Around.** Active claims tend to be narrow. You can sidestep the Mellanox "macro credit" idea by limiting link span or by using rate-based pacing instead of credit aggregation.

6.31 Rethinking Datacenter Management

From ./AE-Specifications/AE-Specifications/sections/Topology.tex

Owners and operators of the network determine the relationships among distributed applications today. Minimum spanning trees, on which all routing is done, are built, and torn down, by *switches*; based on protocols standardized long ago when we first learned how our computers could communicate.

Today's datacenter architects build their infrastructures using two kinds of boxes: *switches* and *servers*.

We refer to all network devices as switches. Those that route at layer 3 are simply layer 3 switches. They connect them using individual cables, which they bundle together to make them convenient to route within and around physical structures. This forms a *centralized* or *decentralized*topology, where the switches become hubs and servers become leaves.

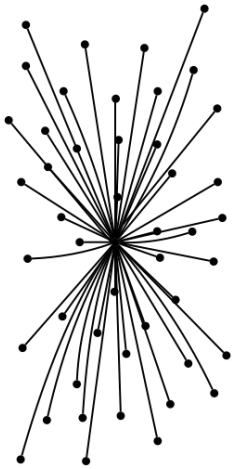


Figure 6.17: CENTRALIZED

Paul Baran's classification provides insight:
`CENTRALIZED (A)` shows 46 univalent nodes connected to a special high radix or *valency*

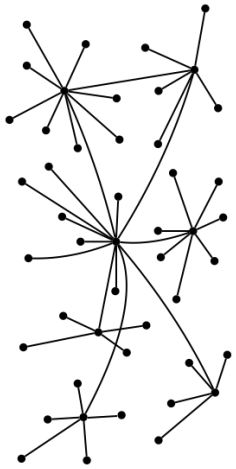


Figure 6.18: DECENTRALIZED

We use `valency` to denote the number of physical ports on a hyper-converged `cell`. A `cell` is a single type of node element (autonomous unit of compute, storage and packet processing).

A `link` is an individual, bidirectional, computation object (an autonomous communication entity between *two* `cells`) This is to distinguish us from radix, used in switches, but is equivalent to degree (δ), in graph theory.

If the central hub dies, all nodes are cut off. `DECENTRALIZED (B)` shows 47 nodes and links, 7 nodes with a valency of 5-7 serve as switches, and 40 as univalent (leaf) nodes. If one of the switches fails, the network fractures into isolated partitions; and only nodes within the partitions can continue to communicate locally. `DISTRIBUTED (C)` shows 47 identical, multivalent (valency ~ 5) nodes, and 98 links. The network has better resilience: failed nodes are routed around, and many links must fail before *any* node is finally isolated.

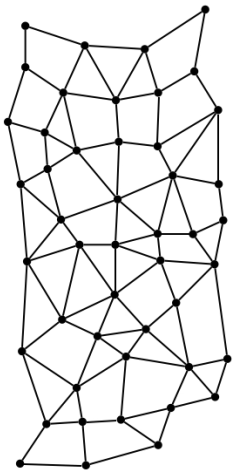


Figure 6.19: DISTRIBUTED

Datacenter Topologies

`CENTRALIZED` topologies are avoided because they represent bottlenecks and have a single point of failure. `DECENTRALIZED` topologies are (hub & spoke) topologies may be economically necessary for large-scale geographically disparate systems like the interstate roadways and airline networks. are most prevalent, which is surprising given how *non-*

optimal they are when looked at from a perspective of distributed microservices container life-cycles

Distributed microservices have an overwhelming predominance of East-West Traffic. Containers can be created in microseconds and last only seconds or milliseconds.

Today's datacenter networks evolved from their roots in the ad-hoc connection of Ethernet broadcast domains with switches housed in wiring closets and managed by individuals with specialized expertise in routing and proprietary management interfaces.

The SPoF's in decentralized topologies are mitigated by redundancy in modern multi-slice Clos Networks. Modern Clos networks typically have two, three or four *slices* of spline and leaf switches, along with multiple sets of cables.. Switches networks that perpetuate this model, are embarrassingly complex, unreliable, arcane, and parochial. This results in very high operational costs, poor security/high vulnerability, and nothing close to five nines reliability [From Joe Howard, The perpetuation of complexity:

> "Ethernet and IP networking is embarrassingly complex, unreliable, arcane, and parochial. That results in very high operational costs, poor security/high vulnerability, and nothing close to five nines reliability. In almost any other product category this would be considered unacceptable. Network technology has changed very little since the late 1980s, with the exception of faster speeds/feeds and some additional protocols and features."

`DISTRIBUTED` topologies are rarely used (so far) in datacenters, except for a few HPC applications. Except for HPC applications, which have used various forms of hypercube routing, based on a cartesian coordinate system of destination addresses (a God's-Eye-View), which assumes that failures are rare, and uses complex band-aids to route around failed nodes.

When we use a *relative* addressing scheme instead, routing around failed nodes becomes far simpler, and we can build entirely new kinds of stacked graph covers to provide functionality not previously needed (or envisaged) on hypercube interconnects..

However, within the same *or lower* capital cost, `DISTRIBUTED` topologies provide: greater resilience, lower latencies, higher available bandwidth and far more flexibility; by connecting `cells`

We combine switches and servers, into a single concept: `cells`, and make them *substitutable (i.e. although they may not be identical in all aspects of their capabilities, they can at least be managed as one 'type'.), which, in turn, makes them* fungible, *and easier to manage. The additional density*

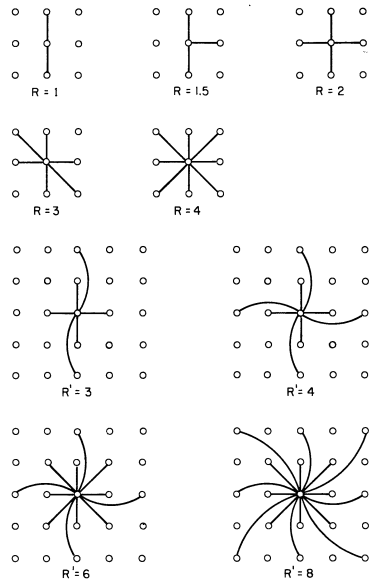


Figure 6.20: Baran: Definition of Redundancy Level

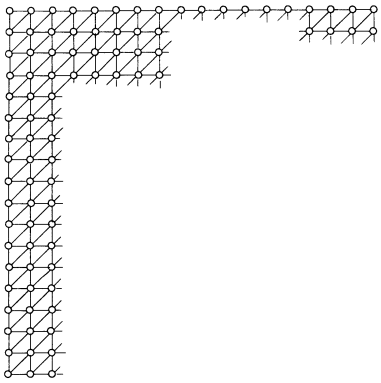


Figure 6.21: Baran: Array of Stations

of the physical topology, afforded by the `cell`*'s 'middle' range valency (*5 *to* 9*), enables far richer virtual topologies to be built.* directly with neighbor to neighbor (N2N) connections rather than through a switched or aggregated network. By not perpetuating the management complexity of switched networks, and introducing new, simpler, control/forwarding planes through `cells`, we can also dramatically lower operational costs.

> Perhaps the time has come to recognize the genius of Paul Baran's insights, and ask why `DISTRIBUTED` topologies are not deployed in datacenters, where their resilience and security can be readily exploited?

Datacenter Programmability

Two types of teams co-evolved to manage modern datacenters: one to design and manage the networks, and one to program and manage the servers. This worked when datacenters had a single owner or tenant, their applications and physical infrastructure evolved slowly, and different business units could work within their own silo's. This is no longer a viable architecture in today's highly dynamic multi-tenant datacenters.

> Distributed applications can no longer afford to be held back by the slow pace of networking innovation.

While programmable switches may be a promising approach to improve the performance and manageability of datacenters, they are still (a) under the control of the network owners and operators, and (b) limited by the low-level endpoint routing and packet forwarding paradigm of today's network engineering. What is needed to complete this revolution is to include the cells (agents on servers) as first class members of this set of devices which are allowed to route packets as well as process them.

TRAPHs: Programmable Application Topologies

Critical layers are missing between applications and infrastructure: a layer which contains the evolving *graph* relationships of modern microservices. A substrate that programmers can own and manage themselves[2]. This provides the missing abstraction for a programmable, and deterministic-when-needed, topologies as tools and resources to the application architect. For example, application programmers can program these TRAPHs (Tree gRAPHs) using a Graph Virtual Machine (GVM) to provide services such as distributed consensus, atomic broadcast, and presence management among members of a cluster or microservice set.

[2] A substrate that can work in conjunction with the simpler forwarding functions within NICs and switches

TRAPHs enable datacenter operators to organize *graphs* of resources, managing them on trees, enabling computing on graphs. From the perspective of different vantage points, each with least-privilage[3]. They also provide developers of microservices complete freedom (within the nodes assigned to them), to programmatically determine their sub-relationships, and the protocol characteristics most needed for their applications.

[3] E.g. Managing realms, jurisdictions, tenants and sub-tenants as graphs instead of lists.

Examples

The advantage or using TRAPHS over a distributed network is that application developers can program their behavior instead of having to wait for permission, or suffer the externalities of the network optimizing itself without regard to the application's health. This simplifies some important use cases such as:

Logical & Virtual Segregation Planes Enable capability-based security graphs, to provide secure containment of communication environments for multi-tenant infrastructures. E.g., exchange the management of lists (ACL's and `iptables`) by replacing them with richer and more manageable `graph equations`. Virtual Segregation Planes: graph applications: erasure coding, machine learning, etc.

Coherent graph overlays where cache *heterarchies* co-exist to automatically manage the placement and eviction of caches based on request patterns. One use case would be a coherent configuration file system, which provide a unified mechanism to keep configuration files synchronized, for Docker, etc. Another would be a *coherent* `memcached`. Eliminating cascade failure incidents, like Google saw recently spread to all regions of their Cloud Platform due race conditions to update configuration files.

Managing Infrastructure as Sets, Graphs & Tensors, instead of *Boxes, Files & Lists.* All routing is predicated on building shortest path trees, e.g. Bellman-Ford for L2, or Dijkstra at L3. The roots for these trees are in the switches, and thus under the administrative control of network owners and operators. With TRAPHs, large subgraphs, or graph-covers comprising `cells` and `links` allocated to a particular tenant, may be used by that tenant for any topology whatsoever, including allocation to sub-tenants. *Graph computing* on TRAPHs (Tree-gRAPHs) enable automatic mapping of the natural DAG relationships of distributed applications on nested datacenter infrastructure resources.

Conclusion: Allowing application developers to build and manage their own routing substrate under API control would dramatically improve the performance, efficiency and flexibility of modern infrastructures, reducing inter-tenant interference, enabling privacy, and improving manageability.

6.32 The Evolution of Baran to Chiplets

6.33 Baran Distributed

6.34 Baran Chiplet

6.35 Transition to Layer 3-Centric Datacenter Design

Modern datacenter network architectures are increasingly gravitating toward Layer 3 (L3) routing as the default mode of operation. While Layer 2 (L2) bridging continues to underpin legacy assumptions in application design, new trends in silicon, open networking, and container orchestration are enabling a scalable, routed foundation across the fabric.

6.35.1 The Fall of Bridging: Application-Layer Assumptions

While the network core transitions to L3, many application-level constructs still implicitly assume bridging:

- **Service and Node Discovery:** Often rely on broadcast, assuming a shared subnet.
- **Cluster Heartbeats:** Use multicast within a broadcast domain.
- **VM Mobility:** Presumes persistent MAC/IP identity and L2 reachability.

These patterns constrain scalability and impede network abstraction.

6.35.2 Historical Impediments to L3 Adoption

Legacy resistance to L3 stemmed from:

- Cost: Vendors historically charged premiums for L3 features and protocols (e.g., BGP).
- Complexity: L3 routing was perceived as difficult to configure.
- Lack of Host Support: Quality routing protocol stacks were unavailable on servers.

These constraints are no longer valid.

6.35.3 New Enablers for Routed Datacenters

- **Merchant Silicon:** Broadcom, Cavium, and Mellanox now offer chips with bridging and routing parity in performance and cost.

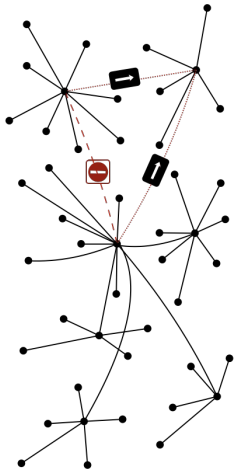


Figure 6.22: Partial Network Partitioning

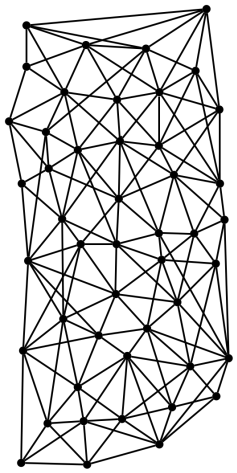


Figure 6.23: Distributed (valency 8)

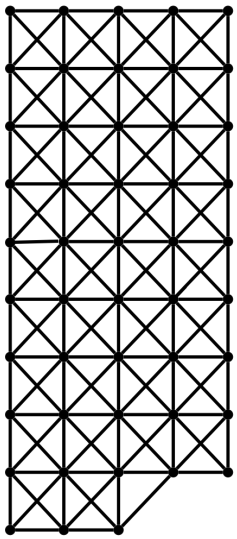


Figure 6.24: Baran Chiplet

- **Open Networking OSes:** Solutions like Cumulus Linux support native routing (e.g., Quagga, BIRD, ExaBGP).
- **Windows Server Support:** Native BGP support since 2012.
- **Routing Protocol Advances:** BGP and OSPF unnumbered simplify configuration.

6.35.4 Kubernetes and the L3 Application Model

Kubernetes abstracts applications from machines, embracing declarative scheduling and management. Key primitives:

- **Pods:** Collections of containers with shared network/storage context.
- **Services:** Logical groups of pods, accessed via virtual IPs.
- **Namespaces:** Logical partitioning for policy and resource isolation.
- **Replication Controllers / Deployments:** Maintain pod count and manage rolling updates.

Networking Model

Kubernetes defines a simple yet powerful networking model:

- Every pod receives a routable IP address.
- Pods can communicate without NAT, even across nodes.
- Eliminates port mapping and collisions.
- Designed to operate over routed L3 fabrics (underlay or overlay).

Overlay Concerns: While overlays (e.g., VXLAN) enable flexibility, they often incur significant performance overhead. Thus, L3-underlay models are gaining traction.

6.35.5 Security and Isolation

Network policy engines (e.g., Calico) leverage namespaces and DAG-style app definitions to:

- Restrict Pod-to-Pod traffic
- Isolate workloads by namespace
- Enforce policy using CNI plugins

6.35.6 Routing on the Host

Hosts can participate in the routed fabric via:

- **BGP Config:**

```
router bgp 65534
  bgp router-id 10.10.1.1
  neighbor eth0 interface remote-as external
  redistribute connected
```

- **Dynamic Peering:** Using BGP unnumbered or listen range.
- **Stub Behavior:** Hosts are always endpoints; never transit.

6.35.7 Services and Load Balancing

- Kube-proxy programs iptables to load balance across pod backends.
- Services expose a stable VIP, accessed via DNS.
- External access enabled via `NodePort` or external load balancers.

6.35.8 Conclusion

The shift from L2 bridging to L3 routing in datacenters is well underway, driven by better silicon, open software, and container-native thinking. Kubernetes exemplifies how scalable infrastructure is no longer tethered to subnet-level assumptions, and instead embraces IP-based identity, network policy, and declarative automation.

6.36 Hybrid Clocks and Common Knowledge

From ./AE-Specifications-ETH/standalone/hlc_common_knowledge.tex

Hybrid clock research, led by S. S. Kulkarni and collaborators, welds a physical-time estimate to a Lamport-style logical counter. The result is a timestamp that (i) respects causality, (ii) stays within a known skew ε of wall time, and (iii) avoids the commit-wait penalties of systems such as Google Spanner. This handout summarizes the key papers, explains how Hybrid Logical Clocks (HLC) operate, and analyzes their impact on the epistemic concept of *common knowledge*.

Table 6.2: Kulkarni hybrid-clock lineage

Year	Work	Key idea
2012	HybridTime	Physical time $\oplus$ counter; bounds skew.
2014	Hybrid Logical Clock	64-bit HLC preserves $e \to f \Rightarrow \mathrm{HLC}(e) < \mathrm{HLC}(f)$.
2015	Hybrid Vector Clock	Vector form; prunes entries older than ε.
16–20	System integrations	HLC in GentleRain+, CausalSpartanX, NuKV.

6.36.1 How Hybrid Logical Clocks Work

Each node maintains

$$\mathrm{HLC} = (\mathrm{pt}, \mathrm{ctr})$$

where `pt` mirrors physical time and `ctr` counts logical ties.

1. **Local event**
   `pt` $\leftarrow$ `now();`
   `pt` $\leftarrow \max($`pt`$,$ `pt`$);$
   `ctr` $\leftarrow$ `(pt==pt) ? 0 : ctr+1;`
2. **Message receive** $(m.\mathbf{HLC})$
   `pt` $\leftarrow$ `now();`
   `pt` $\leftarrow \max($`pt`$,$ m.`pt`$);$
   `ctr` $\leftarrow$ `(pt==m.pt) ?` $\max($`ctr`$,$ m.`ctr`$) + 1$ `: 0;`

   Guarantees:

- If event e happens before f, then $\mathrm{HLC}(e) < \mathrm{HLC}(f)$.
- Absolute error $|\mathrm{pt} - \mathrm{HLC}| \le \varepsilon$ as long as the underlying clock discipline meets that bound.

6.36.2 Epistemic Implications

Classical limit

Halpern and Moses proved that in asynchronous systems absolute common knowledge is impossible; only an infinite tower $E\varphi$, $E\,E\varphi, \ldots$ is attainable.

Hybrid-clock shift

Because every timestamp deviates from real time by at most ε, a node that observes $\mathrm{HLC} \ge T$ can *deduce* that all events with timestamp $< T - \varepsilon$ are in its past. That turns the impossibility into a quantitative statement: facts can become *ε-common knowledge*.

Practical upshot Linearizable commits, causal session guarantees, and wait-free consistent snapshots become feasible after one round trip, provided the skew bound holds.

Limitation If ε blows up (GPS loss, NTP outliers, relativistic links) the deduction fails and classical uncertainty returns. Hybrid clocks do not escape Forward-In-Time-Only thinking; they merely bound its error.

6.37 Relation to FITO Perspective

Your Forward-In-Time-Only critique argues that Newtonian time is a hidden axiom. Hybrid clocks expose—rather than erase—this axiom by making ε explicit. They therefore align with FITO analysis: progress requires either

- shrinking ε (better hardware sync), or
- replacing deterministic bounds with probabilistic or reversible notions of order.

6.38 Open Problems

1. **Dynamic ε.** Adapt clocks when skew drifts.
2. **Probabilistic knowledge.** Treat timestamps as confidence intervals.
3. **ICO-aware clocks.** Design schemes that tolerate indefinite causal order and reversible transactions.
4. **Eventual common knowledge.** Combine HVC pruning with DAG gossip in partitioned networks.

6.39 Takeaways

Hybrid clocks bridge logical causality and imperfect wall time, achieving the effect of common knowledge after a bounded delay ε. They power modern geo-replicated stores without heavy coordination cost, but remain inside the FITO worldview. Future work will loosen or replace the global arrow of time.

References

[1] S. S. Kulkarni and N. Mittal. HybridTime: A decoupling of coordination and time in distributed systems. TR, 2012.

[2] S. S. Kulkarni, et al. Logical Physical Clocks and Consistent Snapshots. 2014.

[3] V. Karmarkar and S. S. Kulkarni. Bounds on Hybrid Vector Clocks. IEEE SRDS, 2015.

[4] H. Wu, et al. CausalSpartanX: Causal consistency over HLC. Middleware, 2016.

[5] J. Su, et al. NuKV: Building a scalable and reliable KV store. SoCC, 2020.

[6] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in distributed environments. JACM 37(3), 1990.

7. Topology

7.1 From Ethernet to Æthernet

It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.1.1 Ethernet was Born

The Original concepts of Ethernet, developed 50 years ago, help us understand the thinking, theoretical concepts, and guide us to practical implementations. It is important to understand the design philosophy, so we can learn the most from the intuition that created the Ethernet revolution. It is instructive to trace the initial intellectual and conceptual steps when Ethernet was first developed, and make sure we are not missing some invaluable intuition.

The original Ethernet was a single coax cable (photon cavity) Half-Duplex 'Bus': alternating between listening and transmitting in 'slots' on the bus.

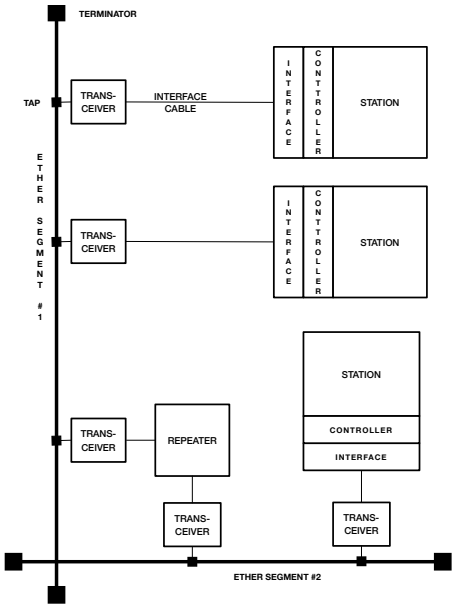


Figure 7.1: Original Ethernet Concepts

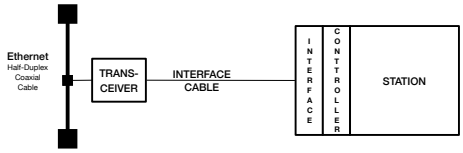


Figure 7.2: Ethernet Components

These temporal 'slots' were time *intervals*: controlled and measured against a local oscillator (often a crystal), which had their own drift and stability characteristics. In a half-duplex world, such as on a single coax cable, this meant that the Transceiver (Transmitter + Receiver) would transmit for half the 'time' and listen for the other half the 'time', and be able to detect collisions (the receiver monitors what it itself is transmitting and compares it to what it is receiving to see.

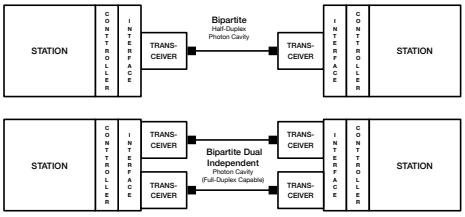


Figure 7.3: Chiplet Æthernet: HDX/FDX

The transmitting station is immediately provided with electrical (signal) feedback which lets it see (a) that it's transmitter was working, and (b) what other receivers might be seeing. From a Shannon perspective, we call this *Initial Information-Feedback (IIF)* and reserve the definition of *Perfect Information Feedback (PIF* for the case where the SerDes at the other end is reflecting what it sees. See section XX for the full theory behind the Dual Back-to-Back (DB2B) Shannon model works.

7.1.2 Ethernet Evolves

Ethernet rapidly evolved to a full duplex situation where there are two separate connections to the media. One in the transmit direction, and

one in the receive direction. However, if one direction is working, and the other direction is not, we could always (in Æthernet) revert back to communicating independently on each cable. This may not have the ideal performance characteristics, but remains a valuable redundancy tool to locally diagnose (and report) unidirectional errors, and flakey fiber connections. This is effectively a dual redundant 'communication' verification tool that can be algorithmically self diagnosing, perhaps in combination to Link training in Modern Ethernet.

7.2 AEthernet Configuration

We begin with two cells, and extend to three. This is the minimum irreducible graph for healing around a broken link. Links can be broken in both directions, or one direction at a time (unidirectional failures)

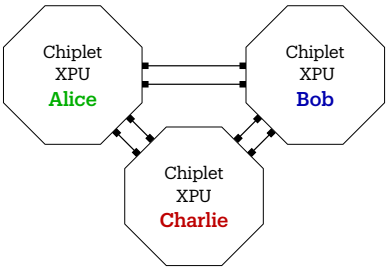


Figure 7.4: Uninitialized XPU Links

7.3 Chiplet XPUs

7.4 Configured Links

We go from Chiplet Servers to something more generalized - XPU's

This 3×3 Tile is the basic fault-tolerant Tile.

Unactivated Links go from being dead to being alive by exchanging configuration packets. These establish the *direction* of the links from

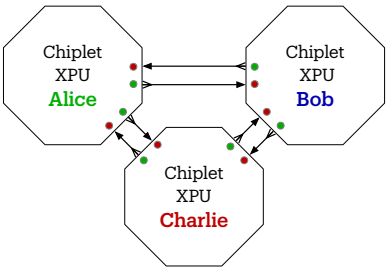


Figure 7.5: Configured Links for TX/RX

7.5 From Repeaters to Switches to Routers, and Back to Repeaters

Bob Metcalfe's concept of a repeater was like what Heaviside discovered – a device to take a fading signal and boost it to recreate a strong signal so as to propagate further. That was the simplistic 'engineers' view of how it worked. The 'physicist's view' was far more interesting – they took a transmission line model, where characteristic impedance $Z_o = \sqrt{\frac{L_o}{C_o}}$.

It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

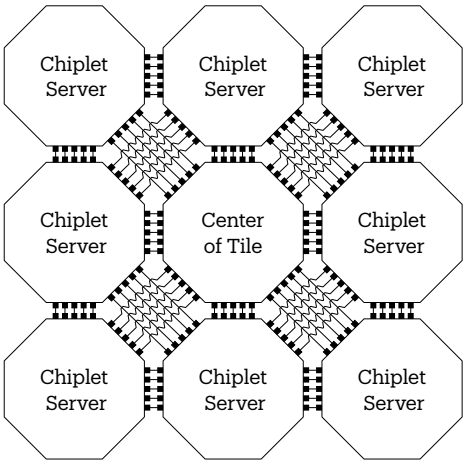


Figure 7.6: 3x3-wire-hop-chiplets

7.6 Minimum Triangle

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing

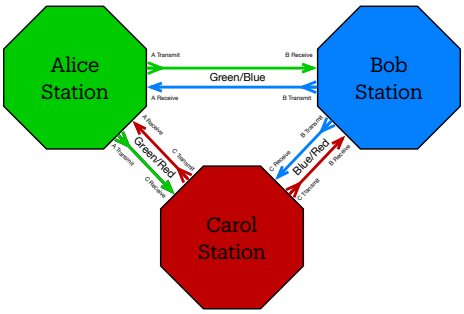


Figure 7.7: Minimum-Triangle.pdf

the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.7 Big Flexible Chiplet

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

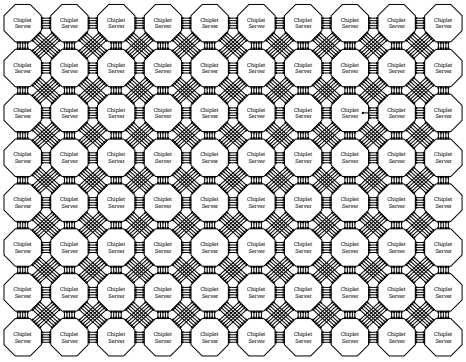


Figure 7.8: 8x10-Chiplets.pdf

7.8 Links are primary communication resources

The link between Alice and Bob is the resource to be shared. What goes over this link is controlled entirely by Alice and Bob.

The GAME the protocol plays ensures *fairness* (50/50 sharing of the link resources). Whatever is left over is offered as a resource to other cells further away.

This removes the need for POLICY – Each link has it's own policy, and can use as much of the bandwidth they need to perform their computation.

Whatever is left over can be offered (advertised) to the rest of the system as a utilization path.

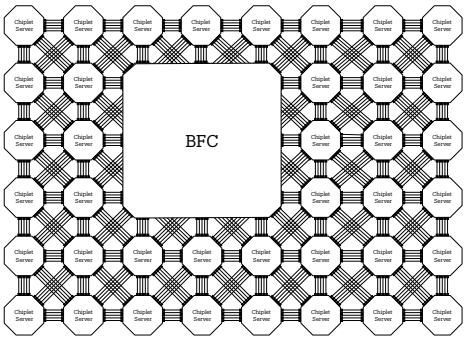


Figure 7.9: Big-Flexible-Chipet.pdf

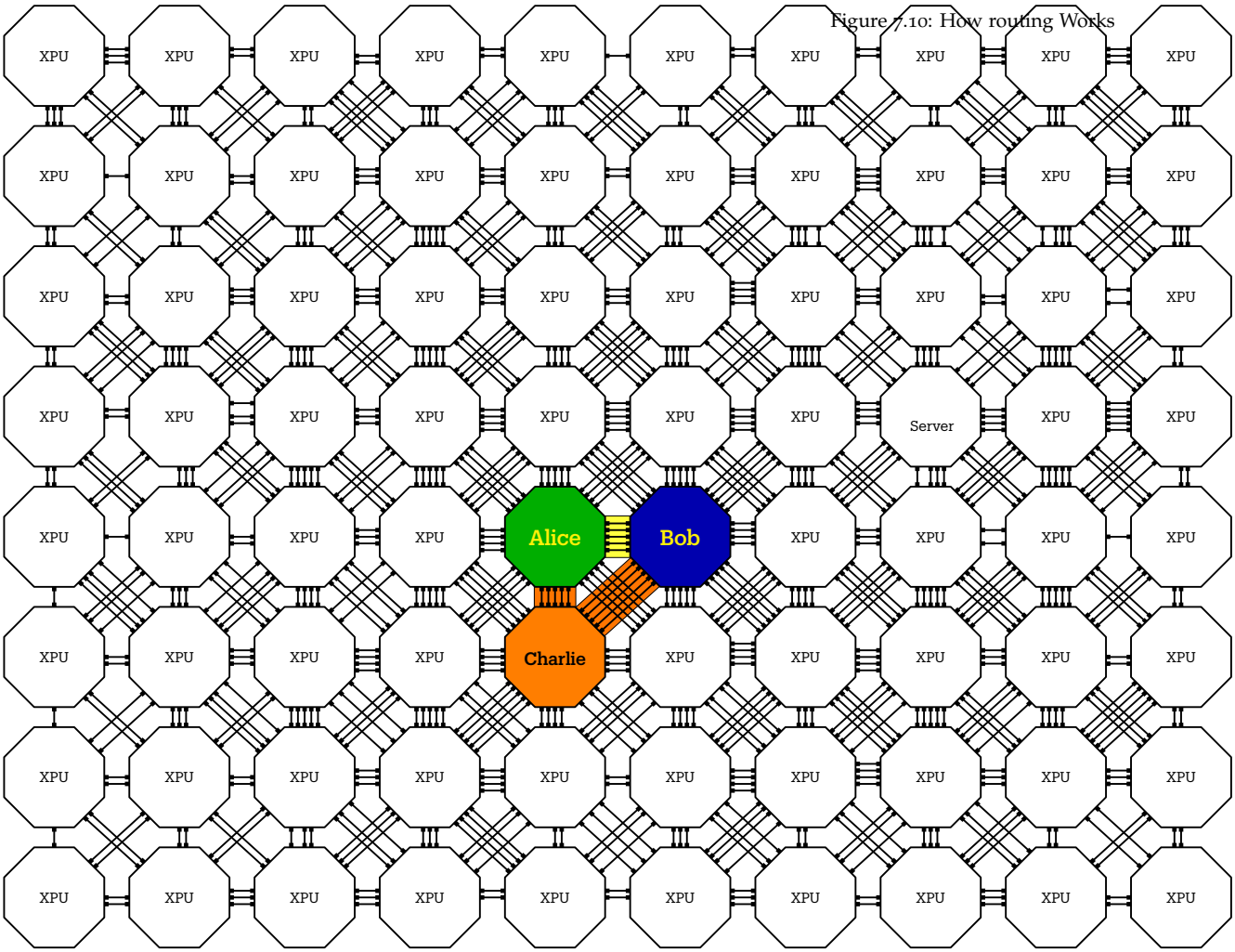


Figure 7.10: How routing Works

7.9 Colored Big Picture

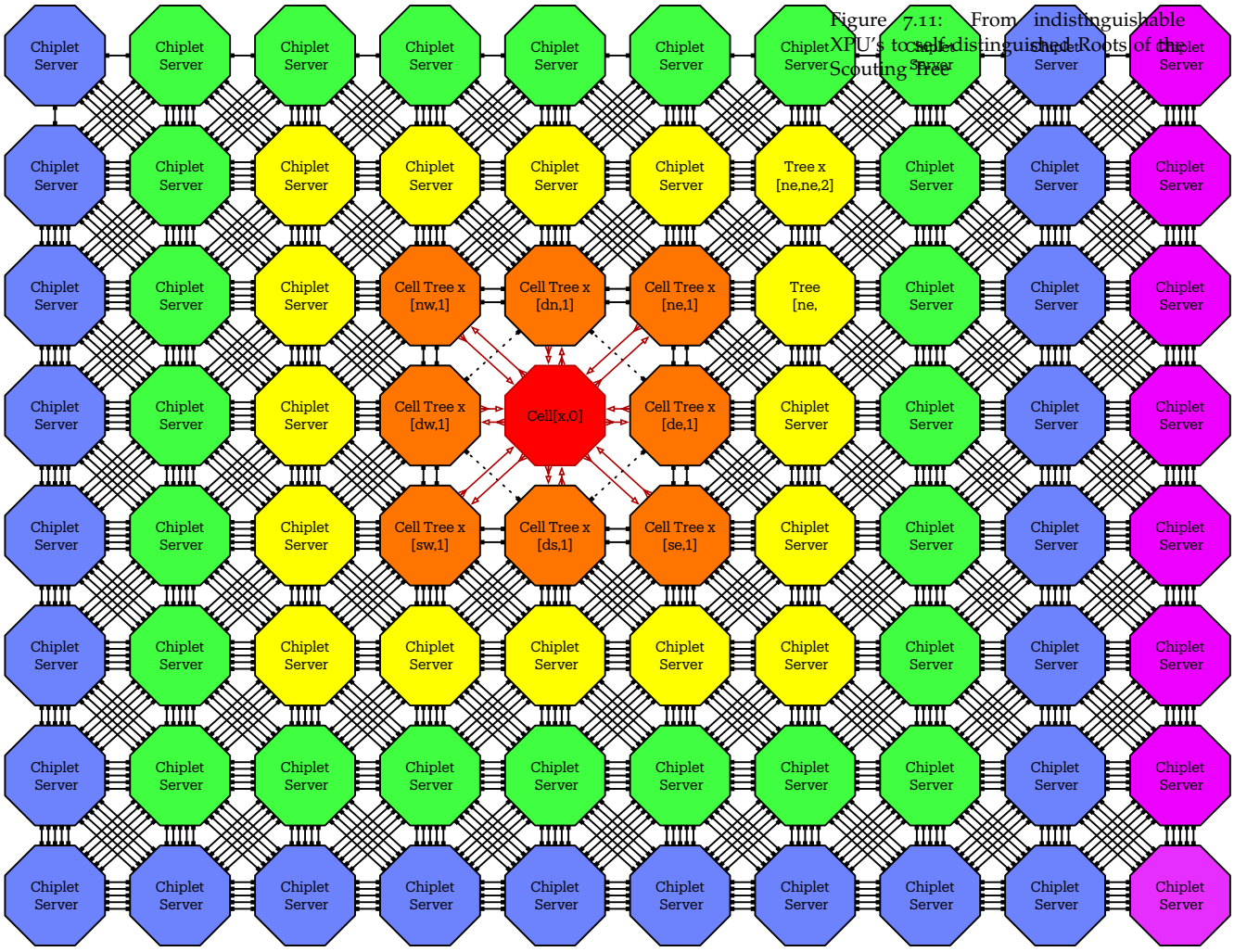


Figure 7.11: From indistinguishable XPU's to self distinguished Roots of the Scouting Tree

7.10 From Physical Cells (XPUs) to Logical Tiles

7.11 Consensus Tiles

1

7.12 Logical Tiles

Logical Tiles are built on top of physical tiles. They have the same 3x3 failure independence characteristics, but help define failure boundaries (the shared fate) of the system. These are discovered, not configured.

Because these

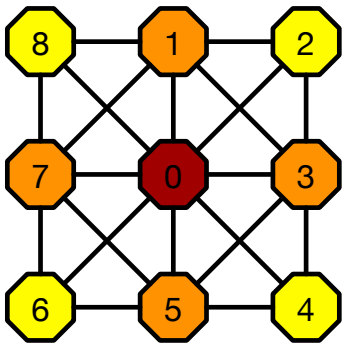


Figure 7.12: Consensus Tiles

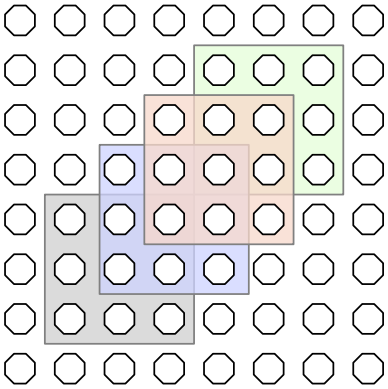


Figure 7.13: 3 x 3 tile

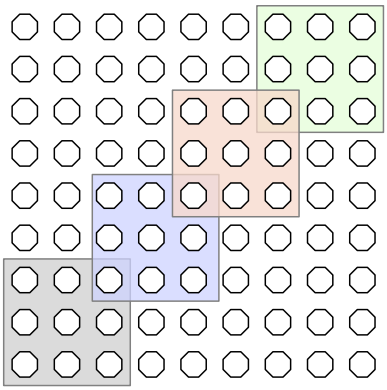


Figure 7.14: 9 x 9 Logical Mesh

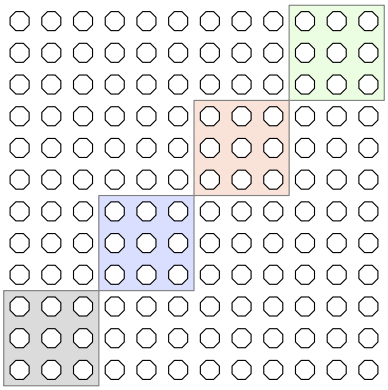


Figure 7.15: 9 x 9 Logical Mesh

7.13 Tiles -2

7.14 Consensus Tiles 3

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.15 Logical Overlays

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.15.1 Unfolded Clos

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing

the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.15.2 Virtual Unfolded Clos

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.16 Physical Overlays

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

7.17 Dragonfly

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

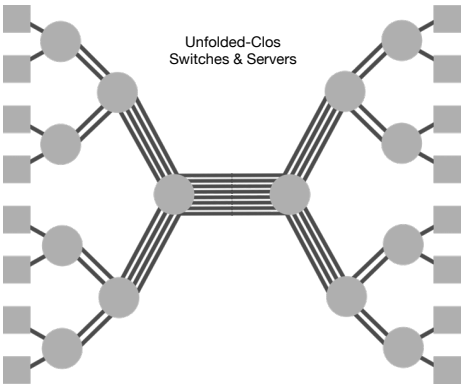


Figure 7.16: Unfolded Clos

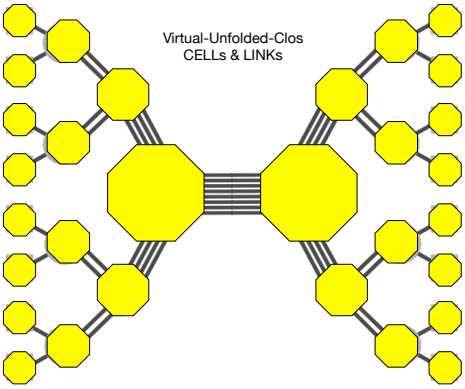


Figure 7.17: Virtual Unfolded Clos. Fish-Eye View of the LINK from any arbitrary LINK

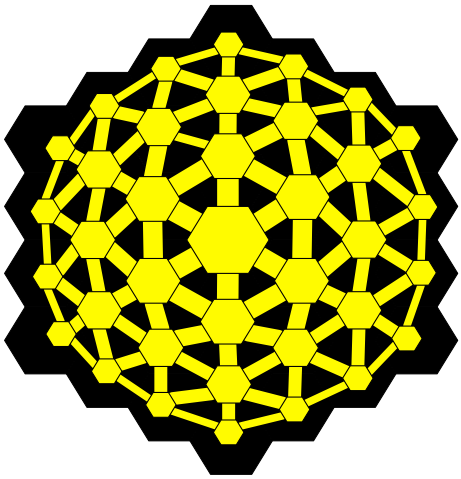


Figure 7.18: Fish-Eye Lens View of the network from an arbitrary Cell

7.18 Graph Aware Determinism

When treating the "Network" as an *opaque cloud*, it's easy to underestimate how varied network partitions when link failures are asymmetrical: A can see B, but B can't see A. In a 4 node setup, there are over 1295 potential partitions, and a flaky network can reproduce them all. From a distributed systems (event ordering in a cluster) as an availability equation, we can easily overestimate how reliable they are, by 3 orders of magnitude.

Link failures are invisible (hidden) in a Clos. They are 100% Visible to us in a local graph of *triangular* relationships.

And that's only the clean (binary) binary failures. Real system *flakey* connections are much worse.

7.18.1 Transactions need a coordinator?

The Æthernet protocol is designed to be exquisitely sensitive to packet loss and corruption We monitor, detect, diagnose link failures, and

recover reversibly and automatically.

7.18.2 A Resilience Metric for Mesh Networks

7.18.3 Graph Laplacian and Algebraic Connectivity

The Graph Laplacian. For a simple, undirected graph $G = (V, E)$ with $n = |V|$ vertices, the *combinatorial Laplacian* matrix L is defined as

$$L \;=\; D \;-\; A,$$

where

- A is the $n \times n$ adjacency matrix, with $A_{ij} = 1$ if there is an edge between i and j, and 0 otherwise,
- D is the $n \times n$ diagonal *degree matrix*, whose diagonal entries are $D_{ii} = \deg(i)$.

The Laplacian L is central in spectral graph theory, encoding many connectivity properties of G.

Algebraic Connectivity (λ_2). Let the eigenvalues of L be ordered as

$$0 \;=\; \lambda_1 \;\leq\; \lambda_2 \;\leq\; \cdots \;\leq\; \lambda_n.$$

The second-smallest eigenvalue, λ_2, is the *algebraic connectivity* (or Fiedler value). It satisfies

- $\lambda_2 > 0$ if and only if G is connected,
- A larger λ_2 generally indicates stronger connectivity and a larger cut is required to disconnect G.

Thus, λ_2 is often seen as a "spectral" measure of how robustly G remains connected under certain disruptions.

7.18.4 Classical Connectivity Measures

Beyond λ_2, there are other classical measures:

1. **Edge Connectivity** $\lambda(G)$: The minimum number of edges whose removal disconnects G.
2. **Vertex Connectivity** $\kappa(G)$: The minimum number of vertices whose removal disconnects G.
3. **Expansion or Isoperimetric Constants**: Relate cut sizes to the cardinalities of sets being separated.

These capture *global* connectivity but may not reflect the incremental or adversarial removal of edges in a constrained-valency network.

7.18.5 Incremental Link Failures in Constrained-Valency Networks

In HPC or data-center systems (e.g. with IPUs or smartNICs), each node has limited valency (e.g. 8 ports), and edges can fail one by one. A single λ_2 value may not capture how partial or progressive failures degrade connectivity.

Why a single λ_2 may not suffice.

- λ_2 is a *one-shot* global measure. It does not directly model how connectivity degrades as edges fail in sequence.
- Some topologies might remain connected but experience severe bottlenecks after a few critical edges fail, which does not show up immediately in a single baseline λ_2.

7.18.6 Potential Approaches for a "Resilience Metric"

7.18.7 Spectral-Based Extensions

(a) Expected λ_2 under random failures. If edges fail independently with probability p, form a random subgraph G_p. One could define:

$$\mathbb{E}\big[\lambda_2(G_p)\big]$$

as a measure of *average* resilience. Larger expected algebraic connectivity implies better tolerance to random edge losses.

Worst-case sequence of λ_2 values. Define:

$$R(k) \;=\; \min_{\substack{F\subseteq E\\ |F|=k}} \lambda_2\big(G-F\big),$$

where $G - F$ is the graph with edges F removed. $R(k)$ measures the smallest λ_2 achievable *after k edge removals*. A graph is more resilient if $R(k)$ remains high for larger k. If $R(k)$ drops to 0, it indicates that with k removed edges, G can be disconnected.

7.18.8 Connectivity-Based Ideas

(a) k-Edge Connectivity Functions. Beyond the single value of $\lambda(G)$ (the edge connectivity), define

$$\phi(k) \;=\; \min_{\substack{F\subseteq E\\ |F|=k}} \Big(\text{size of the largest connected component of } G-F\Big).$$

If $\phi(k)$ remains large, it means removing any k edges fails to isolate more than a small fraction of nodes. This complements λ_2 by focusing on *component sizes*.

(b) Edge-disjoint path counts. Using Menger's Theorem, one can track the number of edge-disjoint paths between certain pairs of nodes. Higher numbers of disjoint paths generally imply more resilient connectivity.

7.18.9 Weighted or Dynamic Laplacian

A *dynamic* Laplacian $L(\mathbf{w})$ might assign weights w_e to edges. If an edge is fully failed, $w_e = 0$. Then one can track how $\lambda_2\big(L(\mathbf{w})\big)$ evolves as edges degrade from weight 1 to weight 0, either in random or adversarial patterns.

7.18.10 A Concrete Proposal

A practical "resilience function" might be:

$$R(k) \;=\; \min_{\substack{F\subseteq E\\ |F|=k}} \lambda_2\big(G-F\big),$$

where the minimum is taken over all subsets F of k edges. Then:

- $R(0) = \lambda_2(G)$ is the baseline algebraic connectivity.
- If $R(k) > 0$, the graph *cannot be disconnected* by removing any k edges.
- The rate at which $R(k)$ decreases with k reflects how fast the network's connectivity deteriorates under incremental failures.

7.18.11 Computational Observations

Exact computation of $R(k)$ can be expensive for large graphs because there are $\binom{|E|}{k}$ subsets. One may:

- Use *heuristics* or *approximation algorithms* to identify critical edges,
- Leverage *min-cut* or *max-flow* bounds to quickly estimate how easy it is to disconnect the graph,
- Perform *sampling* over subsets F if a random measure of resilience suffices.
- The *graph Laplacian* (and in particular λ_2) is a powerful spectral tool. It already gives a measure of connectivity robustness.
- For *incremental* or *adversarial* link failures, a single λ_2 value may not capture the full picture. A *function* $R(k)$ over subsets of size k can indicate how robustly the graph handles multiple simultaneous failures.
- In *constrained-valency* networks, certain edges are more critical, because each node has fewer possible alternate paths. Thus, a spectral-based metric that accounts for edge removals (like $R(k)$) can better reflect real-world vulnerability.
- Combined with classical connectivity measures (e.g. $\lambda(G)$, $\kappa(G)$), a Laplacian-based incremental approach provides a practical, mathematically grounded way to define and quantify *resilience* of a network topology.

7.19 Distributed Systems are Trees on Top of DAGs on Top of Graphs

From ./AE-Specifications-ETH/standalone/Trees-DAGs-Graphs.tex

This essay explores the layered graph-theoretic nature of distributed systems. At the lowest layer, physical and logical interconnects form undirected **graphs**. On top of this lie **DAGs** representing dependency, scheduling, and locking relationships. At the top, application-level consistency and authority are imposed via **trees** such as namespace hierarchies, leadership structures, and commit chains. We further examine how modern datacenters, populated by diverse xPUs (CPUs, GPUs, IPUs, DPUs), break the illusion of shared memory and necessitate protocol designs that exploit the native graph structure using mechanisms such as RDMA.

7.19.1 Graphs: The Physical and Logical Fabric

The physical topology of a datacenter is a graph: nodes represent compute units (CPUs, GPUs, IPUs, etc.) and edges represent communication links (Ethernet, NVLink, InfiniBand, etc.). These links may have diverse properties:

- Bandwidth and latency asymmetries
- Failures or congestion under load
- Scheduled or dynamic routing paths

Unlike the shared memory abstraction, these links form a non-uniform, fault-prone, and inherently asynchronous substrate. Real computation in modern datacenters occurs *on this graph*—not above it.

7.19.2 DAGs: Causality and Locking

On top of the physical graph lies a directed acyclic graph (DAG) representing **causality, scheduling, and consistency constraints**. DAGs arise in:

- **Transaction dependencies**: Operations must follow a directed order to preserve causality.
- **Lock hierarchies**: Preventing deadlock requires acquiring locks in a fixed topological order.
- **Build systems and job schedulers**: Tasks must respect dependencies.

Locking as a DAG

Databases employ lock hierarchies structured as DAGs to prevent circular waits. For example, the following might form a hierarchy:

1. Lock table
2. Then row

3. Then field

Each level narrows scope and follows a partial order. Enforcing that locks are acquired in topological order avoids cycles and hence deadlock.

7.19.3 Trees: Names, Commit Chains, and Leaders

At the top of the stack are trees. These structures are usually logical:

- **Namespace hierarchies**: e.g., file systems, DNS.
- **Leadership trees**: elected leaders per region, rack, or quorum.
- **Consensus and commits**: commit chains or logs form trees (or more precisely, forests with fork resolution).

These trees impose structure on the otherwise messy DAGs and graphs below, enabling:

- Easier authority delegation
- Fault domain containment
- Clear lineage and rollback support

7.19.4 Breaking the Shared Memory Illusion

Shared memory simplifies programming but breaks down in distributed xPU environments:

- Memory isn't uniformly addressable
- Coherence protocols are expensive or infeasible
- Latency variance introduces uncertainty in synchronization

RDMA: Network as Memory Bus

Remote Direct Memory Access (RDMA) partially restores shared memory semantics:

- Allows direct writes/reads between NICs with low latency
- Bypasses kernel and CPU involvement
- Supports zero-copy semantics for performance

But RDMA also forces a shift:

- You must think **asynchronously**
- Buffers must be explicitly registered and tracked
- Failures are explicit, not hidden

7.19.5 Exploiting the Graph: The Path Forward

To fully exploit xPU networks:

- Treat communication paths as first-class citizens
- Build coordination mechanisms that reflect graph topology
- Favor protocols that can adapt dynamically to congestion and partitioning

New system designs should:

1. Replace locking with message-passing wherever feasible

2. Encode application semantics in DAGs, not linear logs
3. Use explicit versioning and conflict resolution mechanisms

7.19.6 Conclusion

Distributed systems are not built on the abstraction of shared memory. They are constructed on a layered composition:

Graphs: physical connectivity

DAGs: causal and logical dependencies

Trees: naming, consensus, and leadership

The challenge of distributed systems is to harmonize these layers while respecting the physical realities of the system. To do so, we must leave behind illusions of synchrony and embrace graph-native programming models.

7.20 Mathematica as a Specification Language

From ./AE-Specifications-ETH/standalone/Mathematica-Spec-Language.tex

Exploring **formally executable specifications** in **datacenter architecture** touches the core of verifiability, reproducibility, and automation in modern systems.

A *formally executable specification* is:

- **Precise and unambiguous:** defined mathematically or via formal syntax.
- **Executable:** interpretable or simulatable.
- **Deterministically testable:** consistent output for consistent input.

Definition: Formally Executable Specification

In datacenter contexts, this implies that hardware, networking, storage, and compute orchestration policies are:

- Executable in simulation or emulation environments,
- Amenable to formal verification for correctness, safety, and performance.

Why It Matters in Datacenters

- **Correctness:** validate failover, routing, and policy enforcement.
- **Optimization:** evaluate configurations automatically.
- **Security:** prove isolation and policy compliance.
- **Confidence:** ensure safe deployment at scale.

Relevant Tools and Technologies

Domain	Tools
Network Architecture	P4, TLA+, NetKAT, Batfish
Storage Systems	TLA+, Ivy, Alloy, Z3 SMT
Orchestration	Kubernetes CRDs, Pulumi, OPA, Nomad
Formal Languages	TLA+, Coq, Lean, Dafny, Alloy
Execution	Mininet, NS-3, OMNeT++, QEMU, Verilator

Figure 7.19: Selected tools for formally modeling datacenter systems

Example: Rack-Aware Topology Specification

Imagine a model with:

- Compute nodes linked via ToR (Top-of-Rack) switches,
- Spine switches in a leaf-spine topology,
- Multi-path routing and QoS,
- VM placement and replication constraints.

The spec could:

- Simulate failures and load distribution,
- Detect routing loops or black holes,
- Evaluate bandwidth and latency guarantees,
- Prove placement constraints meet SLAs.

Vision: "Datacenter-as-Code" Verified

- High-level specs compile into deployable artifacts,
- Every change is property-checked and testable,
- Infrastructure becomes version-controlled logic, replacing spreadsheets and tribal lore.

Evaluating Mathematica for Executable Specification

Mathematica is a powerful computational platform. Its value depends on whether expressiveness or formal rigor is the priority.

7.20.1 Strengths of Mathematica

Limitations Compared to Formal Languages

- **Formal Semantics:** lacks type theory foundations (Coq, Lean).
- **Verification:** no native model checking or invariant proofs.
- **Concurrency:** no Lamport clocks or message-passing models.
- **Determinism:** pattern matching may be nondeterministic.
- **Refinement:** lacks formal spec-to-implementation pathways.

Suitable Use Cases

- Modeling tradeoffs in resource allocation,
- Simulating flows using graph theory,
- Prototyping performance constraints,
- Symbolic scheduling and placement logic,
- Writing executable whitepapers with computation and code.

Where It Falls Short

- Verifying safety and liveness across all states,
- Proving conformance or refinement,
- Modeling concurrency and faults rigorously,
- Integrating with RTL verification pipelines,
- Participating in formal proof communities.

Category	Capability
Symbolic Computation	Excellent for pipelines, graphs, latency models
Executability	Immediate execution and visualization
Expressiveness	Supports discrete, continuous, algebraic models
Rapid Prototyping	Rich in units, semantics, interactivity
Logic Tools	First-order logic, SAT solving, quantifiers
Documentation	Notebooks are self-contained and reproducible

Figure 7.20: Strengths of Mathematica in system modeling

7.20.2 Summary Judgment

Mathematica is:

- **Excellent** for exploratory, high-level modeling and simulation,
- **Weak** for formal verification, proofs, and correctness guarantees,
- **Valuable** as a literate architecture spec tool, but not a full formal methods platform.

7.20.3 Appendix A: TLA+ Model – Rack-Aware Topology

RackAwareSpec.tla

```
---------------------------- MODULE RackAwareSpec ----------------------------
EXTENDS Naturals, Sequences

CONSTANTS Racks, Nodes, Links

VARIABLES rackStatus, linkStatus, trafficMap

(*--algorithm RackAware
variables rackStatus \in [Racks -> {"up", "down"}],
          linkStatus \in [Links -> {"up", "down"}],
          trafficMap \in [Nodes -> [Nodes -> {"ok", "blocked", "reroute"}]];

define
  IsAvailable(n) == \E r \in Racks: rackStatus[r] = "up" /\ n \in Nodes /\ TRUE
end define;

begin
  Init ==
    /\ \A r \in Racks: rackStatus[r] = "up"
    /\ \A l \in Links: linkStatus[l] = "up"
    /\ \A s, d \in Nodes: trafficMap[s][d] = "ok";

  Next ==
    \E r \in Racks:
      /\ rackStatus[r] = "up"
      /\ rackStatus' = [rackStatus EXCEPT ![r] = "down"]
      /\ UNCHANGED <<linkStatus, trafficMap>>
  \/ \E l \in Links:
      /\ linkStatus[l] = "up"
      /\ linkStatus' = [linkStatus EXCEPT ![l] = "down"]
      /\ UNCHANGED <<rackStatus, trafficMap>>;

end algorithm;
=============================================================================
```

Appendix B: Alloy Model – Storage Placement Constraints

StorageModel.als

```
module StorageModel

abstract sig Rack {}
sig Node {
  hostRack: one Rack,
  stores: set Volume
}
sig Volume {
  replicas: some Node
}

fact ReplicationFactor {
  all v: Volume | #v.replicas = 3
}

fact NoReplicaOnSameRack {
  all v: Volume |
    all disj n1, n2: v.replicas |
      n1.hostRack != n2.hostRack
}

pred ShowExample {}

run ShowExample for 3 Rack, 6 Node, 2 Volume
```

7.21 CLOS

From ./AE-Specifications-ETH/sections/Clos.tex

7.21.1 Topology set-up (same 200 servers)

Clos fabric 20 racks, each with 10 servers. Every server now owns **four** NIC ports, all cabled to its top-of-rack (ToR) switch, giving $200 \times 4 = 800$ host links. Each ToR uplinks once to *each* of the four spine switches. A pair of core switches terminates the third level.

8-*regular mesh* The same 200 servers, each equipped with **eight** NIC ports wired into an undirected 8-regular graph. The link count is

$$L_{\text{mesh}} = \frac{200 \times 8}{2} = 800,$$

exactly matching the number of host cables in the Clos system.

7.21.2 Cable inventory

Link class	Clos count	Mesh count
Server–ToR	800	–
ToR–Spine	80	–
Spine–Core	8	–
Server–Server (mesh)	–	800
Total physical links	888	800

Table 7.1: Cable counts after upgrading each Clos server to four NIC ports. The mesh uses the same 800 cables as data-carrying edges, eliminating the 88 upward cables and the entire switch hierarchy above the racks.

7.21.3 Failure-mode magnitude

Treat each link as an independent four-state component $\Sigma = \{00, 01, 10, 11\}$. The number of distinct network states is 4^L, so the number of failure patterns is $4^L - 1$.

$$\log_{10}(4^L) \;=\; 0.60206\, L.$$

Topology	L	Failure modes (order of magnitude)
Clos (4 ports)	888	$\sim 10^{535}$
Mesh (8-regular)	800	$\sim 10^{482}$

Although the Clos now contains more cables, inter-rack traffic is still forced through only 88 uplinks. The mesh distributes both traffic and failure risk across *all* 800 cables.

7.21.4 Path-diversity impact

Clos

- A rack-to-rack flow traverses six vertical hops (Server $\rightarrow$ ToR $\rightarrow$ Spine $\rightarrow$ Core and back down).
- End-to-end success probability is roughly p^6, where p is the per-link health probability.

Mesh

- Every server has eight one-hop neighbours; many multi-hop detours remain even after several failures.
- Loss of one cable only lowers a single server's degree from 8 to 7; global reachability is unaffected.

7.21.5 Key observations

1. **Vertical choke-points remain**. Extra NICs in the Clos enlarge rack bandwidth but do not remove the dependence on 88 spine–core cables.
2. **Risk distribution**. The mesh spreads failure impact evenly; the Clos still concentrates risk in its upper layers.
3. **Equipment footprint**. The mesh eliminates 30 switches (20 ToRs, 4 spines, 2 cores), trading them for denser lateral cabling.
4. **Graceful degradation**. Clos bisection bandwidth falls in 12.5% or 5% steps; mesh capacity decays proportionally to failed cables, with no cliff.

8. Context and Comparisons

8.1 Review of "Synchronous, Asynchronous and Causally Ordered Communication"

8.1.1 Why this paper still matters

- Unifies three delivery disciplines—synchronous, FIFO, and causal—under one axiomatic roof.
- Establishes the strict inclusion chain

$$\text{RSC} \subset \text{CO} \subset \text{FIFO} \subset \text{A},$$

  clarifying what extra guarantees are purchased (and at what cost).
- Introduces the *crown criterion*: a linear-time test to decide whether an execution can be replayed with rendezvous semantics.
- Demonstrates that classic control algorithms (e.g. Dijkstra–Feijen–van Gasteren termination detection) remain safe under the weaker CO model.

8.1.2 Core concepts and results

8.1.3 Model vocabulary

A-computation
: Fully asynchronous; Lamport's three happens-before axioms.

FIFO-computation
: Adds per-channel ordering (send order $\Rightarrow$ receive order).

CO-computation
: Globalises FIFO: if $\text{send}_1 \prec \text{send}_2$ (causally), then $\text{recv}_1 \prec \text{recv}_2$.

RSC-computation
: There exists a *non-separated linear extension* where every send is immediately followed by its receive.

Crown
: Alternating sequence $\langle s_0, r_0, s_1, r_1, \ldots, s_k, r_k \rangle$ forming a dependency cycle; its presence *precludes* synchronous realisation.

8.1.4 Hierarchy (all containments strict)

$$\boxed{\text{RSC}} \subset \boxed{\text{CO}} \subset \boxed{\text{FIFO}} \subset \boxed{\text{A}}$$

8.1.5 Termination-detection case study

CO suffices for the Dijkstra ring-token detector because any basic message can cross *at most one* wave; the colouring rule then guarantees safety.

8.1.6 Implementation guidance (1992 vintage)

- **FIFO**: Per-link sequence numbers plus buffering.
- **CO**: Vector (or matrix) clocks, or handshake I/O buffers that forbid indirect overtakes.
- **RSC**: FIFO + per-message acknowledgement that blocks the sender (classic rendezvous).

8.1.7 Strengths and limitations

Strengths

- Fully axiomatic—no reliance on wall-clock bounds.
- Crown test is linear-time and graph-theoretic.
- Bridges theory and practice with concrete protocols.

Limitations

- Assumes reliable point-to-point channels (no loss, duplication, or Byzantine faults).
- Vector/matrix metadata scales poorly for thousands of nodes; modern systems use compaction.

8.1.8 FITO perspective

FITO Lens: Every step up the hierarchy embeds more irreversible forward-time coupling.

1. **A**: progress driven by timeouts and retries—quintessential Forward-In-Time-Only thinking.
2. **CO**: removes physical-time dependence; ordering relies only on causal DAG, admitting limited reversibility of buffering.
3. **RSC**: rendezvous collapses send/receive into a single spacetime point, eliminating alternative orders.

8.1.9 Comparative note

Lamport (1978)
: Happened-before for totally asynchronous systems.

Birman & Joseph (1987)
: Causal broadcast within groups; present paper generalises to point-to-point and situates it in the hierarchy.

Fidge (1991)
: Vector clocks; adopted here as one implementation route.

8.1.10 Key takeaways

1. Ordering guarantees form a strict lattice; algorithm soundness depends on picking the right rung.
2. "Crown-free $\iff$ synchronisable": if no crowns exist, rendezvous replay is possible.

3. Many rendezvous-based algorithms work under CO, avoiding blocking latency.
4. FIFO alone is *insufficient* for algorithms that assume "no messages in flight."
5. Implementation cost rises sharply: RSC $\rightarrow$ latency; CO $\rightarrow$ metadata; FIFO $\rightarrow$ buffer space.

8.2 Link Fabrics: An Objective Comparison

This document provides a technical overview of five prominent link fabrics: NVLink, UALink, Scale-Up Ethernet, InfiniBand, and conventional Ethernet. We evaluate their architectural characteristics, header overheads, and implications for block-size efficiency, with particular focus on suitability for modern workloads such as AI, HPC, and disaggregated systems. We present side-by-side comparisons of header sizes relative to data payloads and reflect on architectural biases that favor or hinder scalability, latency, and composability.

8.2.1 Introduction

As computation and memory disaggregation evolve, the role of the interconnect becomes central. Whether linking GPUs in an AI training pod, scaling up a symmetric multiprocessor system, or tying together memory and compute pools across racks, the *link fabric* is the substrate on which system performance is built.

Each fabric comes with assumptions about packet size, latency, error recovery, congestion handling, and topology. In this review, we provide a comparative analysis of five contenders:

- **NVLink** (NVIDIA): High-bandwidth, low-latency GPU interconnect
- **UALink** (AMD, Broadcom et al.): Emerging open alternative to NVLink
- **Scale-Up Ethernet**: Evolving conventional Ethernet for tightly coupled systems
- **InfiniBand**: HPC-focused with credit-based flow control and low-latency verbs
- **Conventional Ethernet**: Ubiquitous best-effort packet transport

8.2.2 Architectural Overview

8.2.3 Topology and Purpose

- **NVLink**: Point-to-point or mesh GPU topologies with explicit scheduling and hardware-managed coherence domains.
- **UALink**: Targeted as a broader standard across vendors; switch-based; supports memory pooling and accelerator interconnect.

- **Scale-Up Ethernet**: Designed to bring reliability and ordered delivery to Ethernet via reduced headers, low-latency slicing, and potential for transaction-level acknowledgment.
- **InfiniBand**: Mature switch-based architecture, deeply integrated into RDMA stacks and MPI; emphasis on zero-copy and reliable transport.
- **Ethernet**: Best-effort delivery; scales via oversubscription and buffering; header-heavy; assumes software-managed retry.

8.2.4 Header Overhead vs. Block Size

Link fabrics differ significantly in header-to-payload ratio, especially at small block sizes. Header overhead penalizes small messages in conventional Ethernet, motivating larger minimum block sizes to maintain efficiency. For AI and tightly coupled compute, where atomicity and latency matter, small block sizes with low overhead are preferable.

8.2.5 Header Size vs Block Size Table

Fabric	Header	64B	128B	256B	512B	1024B
NVLink v3	16	25.0%	12.5%	6.3%	3.1%	1.6%
UALink (proj.)	20	31.3%	15.6%	7.8%	3.9%	2.0%
Scale-Up ETH	8–16	12.5%	6.3%	3.1%	1.6%	0.8%
InfiniBand HDR	64	100%	50.0%	25.0%	12.5%	6.3%
Ethernet + IP + TCP	76–92	143.8%	57.8%	28.9%	14.5%	7.1%

Does not Include Atomic Ethernet

We wanted to review and compare these systems before introducing OAE. Imagine what this table looks like when we add OAE's 4-byte header, with fixed size: 64B, 256B, 1024B and 4096B transfers.

8.2.6 Latency and Atomicity Considerations

Atomic operations (e.g., tensor updates, semaphore-based locking) are increasingly being performed across links. The cost of round-trips, retries, or failed speculative execution due to packet drops grows nonlinearly with header size and tail latency variance.

- **NVLink**: High atomicity; limited to GPU domain.
- **UALink**: Claims to enable coherent memory semantics across nodes.
- **InfiniBand**: Explicit verbs for atomic ops, requires RDMA semantics.
- **Ethernet**: Lacks atomic primitives; must emulate via protocols.
- **Scale-Up Ethernet**: Explicit focus on atomic packet slices, with transaction-layer feedback.

8.2.7 Bias Toward Large Packets: A Critical View

Switch-centric fabrics such as Ethernet and InfiniBand often exhibit biases toward large block sizes, due to:

1. **Header amortization**: Larger blocks reduce relative overhead.
2. **Switch buffer economics**: Designed for long flows, not short atomic ops.
3. **Congestion avoidance**: Larger packets allow queue shaping, but penalize small-ops latency.

This introduces architectural bias that disfavors emerging patterns such as sparse updates, fine-grain load/store traffic between heterogeneous xPUs, or distributed execution of transactional graphs.

8.2.8 Conclusion

While Ethernet and InfiniBand continue to evolve, they remain biased toward packetization strategies that penalize small atomic units of work. NVLink and UALink challenge this by focusing on coherence and bandwidth at short distances, but they remain vendor-specific or in flux.

Scale-Up Ethernet presents a new opportunity: to build an atomic-capable, low-latency, congestion-aware transport that preserves Ethernet compatibility while shedding unnecessary biases—especially those that require applications to batch operations just to amortize protocol overhead.

> Future fabrics should not merely transmit data, but should transmit *meaningful, recoverable state*—one slice at a time.

8.3 Ultra Ethernet

8.3.1 Executive overview

Ultra Ethernet Consortium (UEC) is producing an **open, Ethernet-based full-stack specification** targeted at AI and HPC fabrics with one million endpoints, terabit links and sub-microsecond tail latency. Its clean-slate *Ultra Ethernet Transport (UET)* replaces legacy RoCE, adds packet-spray multipath, flexible ordering, congestion-adaptive spraying and hardware collectives.

Compared with InfiniBand, UET keeps the broad Ethernet ecosystem while approaching supercomputer-interconnect performance.

8.3.2 Scope and objectives

- **Scale:** up to 1e6 endpoints and 800;1600 Gbps links.
- **Latency target:** keep 99.9 % of collectives within one-digit µs

- **Profiles:** *AI Base*, *AI Full*, and *HPC*; same wire protocol, differing verb sets.
- **Layer coverage:** Physical, Link, Transport, Software API, Security, Telemetry/Management.

8.3.3 Architecture at a glance

Transport layer – UET

- **Ephemeral connections**: no handshake; state cached and reclaimed per transaction (memory-efficient at scale).
- **Packet spraying** across all equal-cost paths with per-packet sequence numbers; ordering enforced on message completion only.
- **Reliability** uses selective retry plus optional *packet trimming* (header delivered, payload discarded) to signal incipient congestion without head-of-line blocking.

Congestion control

Two complementary schemes:
(1) **Sender-adaptive AI/MD** window (~RTT-speed reaction, ECN-driven).
(2) **Credit-grant** receiver pacing for extreme incast.

Link layer

- **Link-Layer Retry (LLR)** negotiated via extended LLDP; hop-by-hop NACK keeps BER-induced loss from bubbling up.
- **Two traffic classes** prevent deadlock and prioritise control traffic even on PFC-free fabrics.

Software API

Based on `libfabric 2.0` with new verbs: *Deferrable Send*, optimistic rendezvous, expanded atomics and tagged collectives.

Security

Group-keyed AES-GCM; leverages IPSec/PSP primitives yet preserves ephemeral-connection model—important for accelerator offload.

In-network collectives (INC)

Lightweight header extensions allow switches to execute *reduce*, *broadcast* and *all-reduce* without CPU round-trips, standardising what was previously vendor-proprietary.

8.3.4 Key innovations vs. legacy RDMA

8.3.5 Performance and scalability

UEC's internal simulations on Clos-4 fabrics show $>95\,\%$ link utilisation and $<5\,\mu s$ 99.9-percentile all-reduce latency on 4096-GPU clusters

Feature	Ultra Ethernet	RoCEv2 / IB RC
Multipath	Packet-level spray	Flow-hash ECMP
Ordering	Flexible, message-level	Strict in-order
Loss recovery	Selective, trim-aware	Go-back-N
Congestion ctrl.	Auto, zero-tune	DCQCN (manual tune)
Connection state	Ephemeral, pooled	Per-peer, long-lived
Collectives	INC standard	Vendor proprietary / host SW
Security	Built-in AES-GCM	External overlay

Table 8.1: Ultra Ethernet vs. RoCEv2 / InfiniBand RC.

at 800 Gbps. Early silicon demos (Q1 2025) from Broadcom's `BCM57608` NIC confirm wire-rate UET on existing PAM4 links.

8.3.6 Ecosystem readiness

- **Membership**: 90+ vendors incl. AMD, Arista, Broadcom, Cisco, Google, Meta, Microsoft, Intel.
- **Reference code**: open-source UET over commodity NICs, easing migration from libfabric/RDMA apps.
- **Timeline**: v1.0 spec ratification Q3 2025; first commercial gear shipping same quarter.

8.3.7 Gaps and open issues

- **Standardisation overlap** with IEEE 802.1Q priorities, IETF congestion-control drafts and emerging UALink fabric.
- **Inter-op testing** required for packet trimming and INC header parsing across multiple switch ASIC generations.
- **Management model**: YANG/Redfish bindings still draft.

8.3.8 FITO analysis

Conventional Ethernet handles failure strictly forward-in-time: lost packets are re-sent, global ordering re-asserted *after* the fact. UET's *selective trimming* preserves the causal header even under congestion; the payload can be retro-materialised without rolling back link-layer state—an incremental step toward reversible, idempotent communication you favour in Open Atomic Ethernet research.

8.3.9 Conclusion

Ultra Ethernet modernises RDMA semantics for AI/HPC while preserving the economic and tooling advantages of Ethernet. If the consortium delivers on its v1.0 schedule and multi-vendor interoperability, UET could displace proprietary HPC fabrics and close the performance gap with InfiniBand—especially when paired with Broadcom's Scale-Out Ethernet implementation.

8.4 LLC and how it differs from Æthernet

8.4.1 Where LLC Sits in the Stack

```
+-----------------------------+   <- Network layer (IP, CLNP, ...)
|        Network Layer        |
+-------------+---------------+
|     LLC     |      MAC      |   <- IEEE 802 data-link (L2)
|   (802.2)   |    (802.3)    |
+-------------+---------------+
              |
          Physical Layer (L1)
```

The IEEE 802 data-link layer is divided into

- **MAC** (Medium-Access Control, 802.3), which is medium specific and defines framing, addressing, and access rules, and
- **LLC** (Logical Link Control, 802.2), a medium independent sub-layer that offers a uniform service interface to the network layer and, when desired, adds sequencing and acknowledgments.

8.4.2 Core Functions of LLC

Table 8.3: Key LLC features and where similar facilities reside when LLC is not used.

Function	How LLC Provides It	Comparable Mechanism Outside LLC
Service multiplexing	DSAP / SSAP (8 bit each) identify the upper layer protocol.	EtherType field in Ethernet II.
Three service types	Type 1: connectionless, unacknowledged (mandatory). Type 3: connectionless, acknowledged. Type 2: connection oriented with sequencing and flow control.	IP is always connectionless; TCP gives connection oriented reliability above L2.
Media independence	Same LLC PDU rides over 802.3, 802.11, 802.5, FDDI, etc.	Ethernet II framing is Ethernet only.
Optional reliability	HDLC style control field allows SABME, RR/REJ, etc.	Modern networks push reliability to TCP or lossless fabrics such as RoCE PFC.
SNAP extension	Adds 5 bytes (OUI + EtherType) so EtherType based protocols still work over any 802 medium.	Native EtherType in Ethernet II.

8.4.3 Frame-Format Comparison

```
Ethernet II        802.3 + LLC          802.3 + LLC + SNAP
---------------    ---------------      ---------------------
Dest MAC           Dest MAC             Dest MAC
Src MAC            Src MAC              Src MAC
EtherType          Length               Length
                   DSAP                 DSAP = 0xAA
                   SSAP                 SSAP = 0xAA
                   Control              Control = 0x03
                                         OUI (3 B) = 0x000000
                                         EtherType
```

```
Payload ...        Payload ...          Payload ...
FCS                FCS                  FCS
```

Ethernet II places the EtherType immediately after the source MAC and is the framing used by practically all modern IP traffic.[1] With 802.3 + LLC, the receiver must parse the LLC header (DSAP / SSAP) to learn which upper layer service is carried. SNAP keeps the 802 structure while still transporting traditional EtherType values.

[1] See IEEE Std 802.3-2022, Section 3.1.

8.4.4 Why LLC Faded on Ethernet

1. **Simplicity and cost.** Early NICs already spoke Ethernet II, so adding LLC parsing logic gave little benefit.
2. **IP dominance.** Most traffic needed only EtherType 0x0800 (IPv4) or 0x86DD (IPv6), so DSAP / SSAP were redundant.
3. **Redundant reliability.** LLC Type 2 and 3 capabilities overlapped with TCP end to end guarantees.
4. **VLAN tagging.** 802.1Q inserts its own 4 byte shim but preserves EtherType demultiplexing.

As a result, modern NICs understand LLC/SNAP for legacy frames (for example STP, LLDP), yet more than 99 percent of everyday traffic uses Ethernet II framing.

8.4.5 LLC Versus Other Ethernet Related Layers

Table 8.3: Comparative position of LLC and other Ethernet related layers.

Aspect	LLC (802.2)	Ethernet II	MAC Control / PFC
Purpose	Uniform service, optional reliability	Minimal frame wrapper	Flow control, security
Media scope	Any IEEE 802 medium	Ethernet only	Ethernet only
Header size	3 B (8 B with SNAP)	none beyond MAC + EtherType	Control frames are separate 64 B PDUs
Error handling	Optional ACK / REJ at L2	None	Pause frames stop the transmitter; no ARQ
Typical use today	STP, LLDP, some industrial stacks	IP, ARP, VLANs, nearly all data traffic	Datacenter congestion control, MACsec

8.4.6 Take Aways for New Protocol Design

- If you need to multiplex a new L3 protocol, registering an EtherType or using OUI + SNAP is simpler than reviving DSAP / SSAP values.
- Link layer reliability costs latency. Modern reversible or causal ordering schemes are better placed above the MAC, much as RoCEv2 rides over UDP/IP.
- For designs that must traverse Wi Fi or other IEEE 802 media, SNAP framing keeps you inside the standard while preserving familiar EtherType semantics.

8.5 Infiniband vs Ethernet

From ./AE-Specifications-ETH/Infiniband.tex

8.5.1 High-Level Overview

At a high level, **Ethernet** and **InfiniBand** both provide packet-switched networking, yet they originate from very different design philosophies and ecosystems. The widespread perception that **InfiniBand is more reliable** arises from the architectural and operational differences examined below.

8.5.2 Origins and Design Philosophy

- **Ethernet** evolved as a best-effort LAN. Reliability, ordering, and latency guarantees were relegated to higher layers (TCP/IP), Quality of Service, or RDMA over Converged Ethernet (RoCE).
- **InfiniBand** was conceived for high-performance computing (HPC) and modern data-center fabrics, prioritising low latency, high throughput, lossless delivery, and *built-in* reliability.

8.5.3 Transport and Reliability Model

- **Ethernet (traditional)** is best-effort; packets may be dropped, duplicated, or reordered. Higher layers must restore reliability.
- **InfiniBand** embeds a reliable transport protocol in hardware, handling acknowledgments, retransmissions, and flow control with minimal host-CPU involvement.

8.5.4 Congestion and Flow Control

- **Ethernet**: early 802.3 had none. Data-Center Bridging (DCB) and Priority Flow Control (PFC) improve matters but add complexity and are not always enabled end-to-end.
- **InfiniBand**: credit-based flow control and end-to-end congestion management prevent buffer overruns without dropping traffic.

8.5.5 Packet-Loss Behaviour

- **Ethernet**: under load or misconfiguration, packets drop and TCP must recover, causing latency spikes.
- **InfiniBand**: packets are rarely dropped; congestion produces backpressure instead.

8.5.6 Latency and Jitter

- **Ethernet**: latency and jitter depend on traffic, buffering, and TCP recovery.
- **InfiniBand**: microsecond-scale latency and very low jitter via lightweight stack, hardware offload, and zero-copy RDMA.

8.5.7 RDMA Support

- **Ethernet**: RoCE provides RDMA but demands a tuned, lossless fabric (PFC, ECN, buffer sizing), often fragile and vendor-specific.
- **InfiniBand**: native RDMA with hardware reliability and no special tuning.

8.5.8 CPU Overhead

- **Ethernet**: full TCP/IP stack consumes CPU unless offloaded by SmartNICs.
- **InfiniBand**: host-channel adapters offload packetisation, ordering, and reliability, yielding lower CPU utilisation.

8.5.9 Ecosystem and Deployment

- **Ethernet**: ubiquitous, inexpensive, multi-vendor, and scaling to 800 Gb/s.
- **InfiniBand**: dominant in latency-sensitive HPC/AI but niche, costlier, and largely single-vendor (NVIDIA/Mellanox).

8.5.10 Summary Table

Feature	Ethernet	InfiniBand
Reliability	Best-effort (TCP)	Hardware-enforced
Latency	Milli-second typical	Micro-second
Jitter	High	Very low
Congestion control	Optional (DCB/PFC)	Built-in
Packet loss	Possible	Avoided by design
RDMA	RoCE (complex)	Native
CPU overhead	High (software stack)	Low (offload)
Primary use	General networking	HPC / AI clusters

Table 8.4: High-level comparison of Ethernet and InfiniBand.

8.5.11 RoCE: RDMA over Converged Ethernet

8.5.12 Variants

- **RoCE v1**: Layer 2 only, not routable.
- **RoCE v2**: UDP/Layer 3, IP-routable.

8.5.13 Why Reliable Deployment is Hard

1. **Lossless Fabric Requirement**: depends on PFC; mis-tunes cause deadlocks and head-of-line blocking.
2. **UDP Transport**: inherits best-effort IP semantics unless the fabric is tightly managed.
3. **Ecosystem Coordination**: NICs, drivers, libraries (`libibverbs`), and applications must align or fall back to TCP.

8.5.14 Why Ethernet Remains Dominant

- **Cost & Ubiquity**: every datacenter already runs Ethernet; hardware is commoditized.
- **Interoperability**: multi-vendor openness avoids lock-in.
- **Performance Road-map**: speeds have risen from 10 Gb/s to 800 Gb/s.
- **Software Ecosystem**: the global Internet stack assumes Ethernet/TCP.
- **RoCE as Bridge**: hyperscalers deploy RoCE successfully by exerting strict control over their fabrics.

8.5.15 Conclusion

InfiniBand remains the gold standard for ultra-low-latency, highly reliable HPC workloads. Ethernet is closing the gap through RoCE, SmartNIC offloads, DCB/PFC, and ever-faster links. Its dominance stems from universality and cost, not intrinsic technical superiority.

8.5.16 Design Goals for a Next-Generation Fabric

1. Sub-microsecond latency with deterministic throughput.
2. Hardware-enforced reliability (acknowledgment & retransmission in silicon).
3. *RDMA-first* semantics: zero-copy `PUT`, `GET`, and atomic operations.
4. Programmability: P4/eBPF pipelines in NICs and switches.
5. Security by design: cryptographic authN/authZ and fine-grained access control.
6. Clock-agnostic operation: causal or reversible timing models.
7. Composable transports: reliable/unreliable, ordered/unordered as required.
8. Multi-tenant virtual fabrics on shared hardware.

8.5.17 Key Building Blocks

- **SmartNICs**: onboard CPUs or FPGAs for protocol state, reversibility, and EPI/ONT registers.
- **Flow-Aware Non-Switch Fabric**: dynamic, congestion-aware path scheduling.
- **Unified Declarative Transport**: intent-based API replacing TCP/IP or InfiniBand verbs.
- **Fabric-Wide Memory Space**: global RDMA address space with capability-based security.

8.5.18 Design Framework

Something Old Knowledge == captured information

Something New RED == Information surprisal (the answer to a yes/no question)

Something Borrowed == Tie-in to Rust model (for RPC - Alice owns but Bob borrows)

Something Blue Semantics (Meaning) – The SmartNIC/IPU understands the context

Something Green OCP Green for Open Syntax – goes over PCIe.

From ./AE-Specifications-ETH/Infiniband.tex

At a high level, Ethernet and InfiniBand serve similar roles as network technologies, but they come from very different design philosophies and ecosystems. The perception (and often the reality) that InfiniBand is more reliable than Ethernet stems from multiple architectural and operational differences:

Origins and Design Philosophy Ethernet evolved from a best-effort, general-purpose LAN protocol for office use. Reliability, ordering, and latency guarantees were added later (e.g., via TCP/IP, QoS, or RDMA over Converged Ethernet—RoCE). InfiniBand was designed from the start for high-performance computing (HPC) and data centers, prioritizing low latency, high throughput, lossless transmission, and reliability at the transport layer.

Transport and Reliability Model Ethernet (traditional) is best-effort. Packets may be dropped, duplicated, or arrive out of order. It's the responsibility of higher layers (e.g., TCP) to ensure reliability. InfiniBand includes a reliable transport protocol in hardware, supporting retransmissions, acknowledgments, and flow control natively, without software stack involvement.

Congestion and Flow Control Ethernet historically lacked built-in congestion control (early 802.3 Ethernet had none). Modern enhancements like Data Center Bridging (DCB) and Priority Flow Control (PFC) try to fix this, but they're complex and not universally deployed. InfiniBand uses credit-based flow control and end-to-end congestion control, ensuring no buffer overruns and graceful behavior under load.

Packet Loss Behavior Ethernet (especially under load or with improper configuration) will drop packets, which TCP must recover from. In large-scale systems, dropped packets can cause latency spikes or retries. InfiniBand virtually never drops packets under normal operation. When congestion happens, packets are backpressured rather than discarded.

Latency and Jitter Ethernet is not inherently low-latency. Latency and jitter vary depending on traffic conditions, switch buffering, and TCP behavior. InfiniBand achieves microsecond-scale latencies with

very low jitter, due to lightweight protocol stack, hardware offloads, and zero-copy RDMA support.

RDMA Support Ethernet supports RDMA via RoCE, but it requires very careful network tuning (lossless fabric, PFC, ECN, etc.), which is brittle and often vendor-specific. InfiniBand natively supports RDMA with hardware reliability, requiring no tuning or lossless Ethernet tricks.

CPU Overhead Ethernet (via TCP/IP stack) incurs significant CPU overhead unless you're using SmartNICs or offload engines. InfiniBand offloads nearly everything—packetization, reliability, ordering—to the HCA (Host Channel Adapter), allowing much lower CPU utilization.

Ecosystem and Deployment Ethernet is everywhere—ubiquitous, cheap, interoperable, and increasingly faster (100/200/400/800 Gbps). InfiniBand is dominant in HPC and AI clusters where latency and throughput are critical, but it's niche, expensive, and vendor-constrained (mainly NVIDIA/Mellanox).

Let's start by exploring RoCE (RDMA over Converged Ethernet), then dig into why Ethernet dominates, despite its technical inferiority in some contexts.

8.5.19 1. What is RoCE?

RoCE stands for RDMA over Converged Ethernet, and it attempts to bring InfiniBand-style performance (especially RDMA) to the Ethernet world. It's driven largely by:

The widespread dominance of Ethernet hardware. The desire to reduce CPU overhead using zero-copy RDMA. Avoiding the proprietary nature and cost of InfiniBand gear. Variants:

RoCE v1: Works at Layer 2. Not routable—same Ethernet broadcast domain.

RoCE v2: Works at Layer 3. IP routable (UDP-based).

8.5.20 2. Why RoCE is Hard to Deploy Reliably

RoCE does not bring InfiniBand-level reliability automatically. Here's why:

- a. Requires Lossless Ethernet Fabric
  RoCE depends on PFC (Priority Flow Control) to avoid packet loss, because it assumes a reliable transport. But PFC is fragile and can cause deadlocks, head-of-line blocking, or cascading failures if mis-

configured. Engineers must carefully tune every switch for flow control, ECN, buffer sizing—an operational burden.

- b. RoCE v2 is UDP-based While it's routable, it inherits none of the guarantees that InfiniBand hardware provides. It relies on best-effort IP routing, unless the fabric is tightly managed.
- c. RDMA Requires Ecosystem Support You need NICs, drivers, libraries (e.g. libibverbs), and applications that support RDMA. Without full-stack coordination, fallbacks to TCP/IP hurt performance.

8.5.21 3. Why Ethernet Remains Dominant

Despite these shortcomings, Ethernet continues to dominate outside of niche HPC environments. Here's why:

- a. Ubiquity and Cost Every datacenter already runs Ethernet. It's cheap, commoditized, and standard. Hardware (switches, NICs) is available from dozens of vendors.
- b. Interoperability Ethernet is vendor-agnostic. InfiniBand is dominated by NVIDIA/Mellanox, leading to vendor lock-in.
- c. Evolving Performance Ethernet speeds have scaled up dramatically: 10/25/50/100/200/400/800 Gbps. With SmartNICs and programmable switches, it's closing the gap.
- d. Software Ecosystem The entire Internet stack runs on Ethernet/TCP/IP. Switching is hard. Tooling, monitoring, and network engineering knowledge is Ethernet-centric.
- e. RoCE as a Bridge While complex, RoCE offers a path to InfiniBand-like performance while preserving Ethernet investment. Large-scale cloud providers (e.g., Azure) have successfully deployed RoCE with tight control over the fabric. Conclusion:

InfiniBand is still the gold standard for ultra-low-latency, high-reliability HPC.

Ethernet, while historically less reliable, is catching up via:

RoCE (with caveats),

SmartNICs (e.g., BlueField, Pensando, etc.),

DCB and PFC tuning,

Fast software stacks (e.g., DPDK, eBPF).

Ethernet's dominance is not because it's better—it's because it's "good enough" and universal.

Would you like to dive deeper into SmartNICs or RDMA performance tuning? Or perhaps explore what an ideal next-gen fabric could look like, combining the best of both worlds?

8.5.22 Summary Table

Feature Ethernet (traditional) InfiniBand

Reliability Best-effort (via TCP) Hardware-enforced

Latency Millisecond-scale (typical) Microsecond-scale

Jitter High Very low

Congestion Control Optional (DCB/PFC) Built-in

Packet Loss Possible Avoided by design

RDMA Support RoCE (complex) Native

CPU Overhead High (software stack) Low (hardware offload)

Use Case General networking HPC, AI, low-latency clusters

9. History

9.1 Intro

9.1.1 The End-To-End (E2E) Principle is a Failed Architectural Theory

Question: You wrote in your paper that the "E2E principle assumes smart endpoints and a dumb network, which worked when endpoints could coordinate state easily in one core system."

I had long discussions about the E2E Principle with Saltzer and David Clark. We actually discussed this principle in a panel I moderated in a conference in Dubai last Feb.

Saltzer's input:

"Because there may be trade-offs among competing considerations, we called end-to-end an 'argument' rather than proposing that it be a hard-and-fast design rule. If we were writing the paper today, it would undoubtedly include some discussion of recent 'computing in the network' concepts, and point out the ways that at least some of those concepts are consistent with an end-to-end, application knows best, perspective."

"The basis of the end-to-end principle is that the application knows best. If the application has the ability to tell an in-network service 'Do X when you see my packets' that would seem to support the end-to-end principle."

Answer: The end-to-end principle is designed for file transfer, not transactions.

When network designers believe they have a god-given right to Drop, Reorder, Duplicate and Delay packets, this creates unbounded reordering buffer resource explosions on the endpoints. Applications are forced into only one solution: **Timeout and Retry (TAR)**—the root of all evil.

This is fundamentally in conflict with what modern applications need for ACID guarantees.

9.1.2 Background and Pre-reading

Before evaluating these technical proposals, it helps if the reader has a solid background in the theory and practice of networking.

We recommend The excellent Books by Larry Peterson and Bruce Davie: Computer Networks: A Systems Approach, and other books

in their series, particularly TCP Congestion Control: A Systems Approach. Without this as a background, it would be easy to think that the proposals you see in this document are naïve.

While we will try to use concepts, definitions and literature that are familiar to the Computer and Networking Community, if that was all we did we would be trapped in the valley of incrementalism.

So there are new concepts and terminology, often derived from other branches of physics and engineering. We will try to introduce them as clearly as we can, but the reader is recommended to have their own therapy session with their favorite AI when they find themselves bored, angry or overwhelmed.

As far as these new concepts are concerned, we know we won't bring all of you along. But we ask you to at least try before unceremoniously reject what you see here.

A special essay has been written for those who just want to hunker down and stay in their silo of knowledge: The Network Warrior.

9.2 ALOHA

This section compares the classical ALOHA protocol with its improved variant, Slotted ALOHA, highlighting differences in their design, collision behavior, and efficiency.

9.2.1 Introduction

The **ALOHA protocol**[1] allows nodes to transmit packets at any time, leading to frequent collisions. **Slotted ALOHA**[2], in contrast, divides time into discrete slots, allowing transmissions only at the beginning of a slot, thereby reducing the chance of collisions.

[1] Developed in the late 1960s at the University of Hawaii for radio communications

[2] Introduced shortly after by Roberts in 1972

Abramson (1970); Kleinrock and Tobagi (1975)

9.2.2 Comparison Table

Feature	ALOHA	Slotted ALOHA
Time structure	Any time	Slot-aligned
Collision probability	High	Lower
Efficiency (max)	$\approx 18\%$ $(1/2e)$	$\approx 37\%$ $(1/e)$
Implementation complexity	Simple	Needs synchronization
Analogy	Random shouting	Timed shouting

Table 9.1: Key differences between ALOHA and Slotted ALOHA.

9.2.3 Visual Comparison

Figure 9.1 shows a visual comparison of packet transmissions and collisions in both protocols.

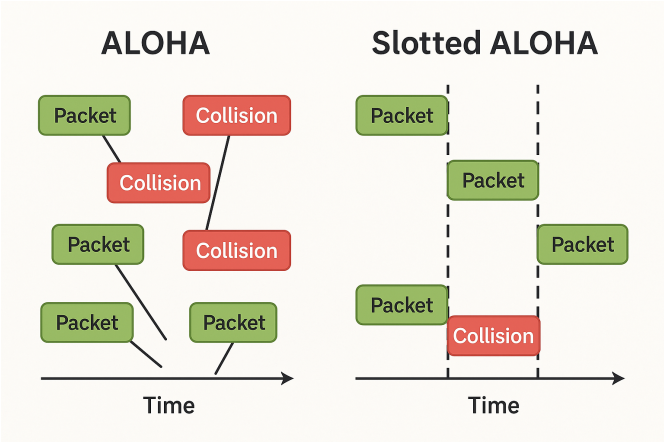


Figure 9.1: Collision behavior in ALOHA (left) vs. Slotted ALOHA (right).

9.2.4 Summary with Sidenotes

ALOHA allows nodes to transmit without coordination,[3] but suffers from a high collision probability.

Slotted ALOHA improves throughput by enforcing transmission only at predefined time slots,[4] achieving almost twice the maximum throughput.

Overall, Slotted ALOHA introduces a modest increase in complexity but greatly improves network performance in high-load scenarios.[5]

[3] This freedom results in many partial collisions where packets overlap partially in time.

[4] By waiting until the next time slot boundary, nodes avoid partial overlaps.

[5] Especially important for early satellite and Ethernet networks.

9.3 ATM

In the early 1990s, the ATM Forum became the battleground for a pivotal debate in networking: how to manage congestion in a cell-based fabric designed to unify voice, video, and data traffic. The two main contenders were **Hop-by-Hop Flow Control** and **Rate-Based Flow Control**. Each represented a fundamentally different view of how best to achieve performance guarantees and fairness across a heterogeneous, multi-hop network composed of 53-byte cells.

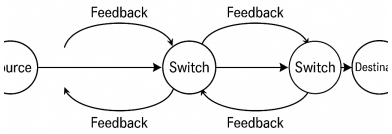
9.3.1 The Challenge: ATM's Dual Mandate

ATM was envisioned as the unifying transport for all digital communication, requiring it to offer both the deterministic timing of circuit-switched networks and the efficiency of statistical multiplexing. This meant that congestion control was not merely a performance tweak, but a contractual necessity to maintain promised QoS levels.

ATM's cut-through switching and small fixed-size cells eliminated much of the buffering flexibility available to IP networks. It had to prevent congestion, not recover from it.

9.3.2 Hop-by-Hop Flow Control

Hop-by-hop flow control works by applying *local backpressure*: each switch monitors its output buffers and signals its upstream neighbor to slow or stop traffic as congestion builds.



op-by-hop flow control: feedback loops are short but
ropagate slowly end-to-end.

Figure 9.2: Hop-by-hop flow control: feedback loops are short but propagate slowly end-to-end.

Advantages

- Immediate local reaction to congestion.
- Fine-grained control over buffer occupancy.
- Simple logic for small or low-diameter networks.

Drawbacks

- Scalability concerns: lacks consistent end-to-end semantics.
- Head-of-line blocking and poor latency propagation in long paths.
- Fragile under path diversity and route reconfiguration.

9.3.3 Rate-Based Flow Control

Rate-based flow control, standardized as part of the ATM ABR (Available Bit Rate) service class, aimed to regulate traffic from the *edge*. Sources declare a desired transmission rate, and switches generate **Resource Management (RM)** cells containing congestion feedback. These RM cells traverse the path forward and backward, carrying fields such as *Explicit Rate (ER)* that guide sender behavior.

Advantages

- End-to-end perspective scales better with network size.
- Enables policy-driven traffic contracts and rate shaping.
- Compatible with QoS-aware routing and admission control.

Drawbacks

- More complex per-switch logic and state maintenance.
- Relies on timely and reliable RM cell feedback.
- Convergence time can be slow in bursty or highly dynamic conditions.

9.3.4 The Verdict: Standardization

After extensive debate, the ATM Forum chose **Rate-Based Flow Control** as the official standard. This decision reflected a belief in the *end-to-end model* of networking, better alignment with telco administration, and superior support for SLAs (Service Level Agreements). Switch

vendors also favored rate-based schemes for their reduced buffer requirements and predictability.

9.3.5 Legacy and Modern Echoes

Although ATM faded from prominence, the ideas from its flow control debate echo through modern networking:

- **InfiniBand** and **credit-based Ethernet** revived hop-by-hop flow control for low-latency datacenter fabrics.
- **TCP Vegas** and **XCP** extended the rate-based idea into congestion-aware transport.
- **PFC** and **QCN** in Data Center Bridging (DCB) illustrate hybrid approaches that combine both paradigms.

In hindsight, both models have value—hop-by-hop for tight fabrics with predictable topology; rate-based for scalable, heterogeneous systems. The ATM Forum chose well for its assumptions—but the future fragmented.

9.3.6 Conclusion

The ATM Forum's choice to standardize rate-based flow control was less a dismissal of hop-by-hop than a reflection of the broader ambitions of the ATM architecture. It aimed to build a global, carrier-grade network substrate. In contrast, datacenters—where predictability and tight control dominate—would later rediscover the strengths of localized flow control.

Rate-based flow control won the standard. But hop-by-hop flow control won the datacenter.

9.4 Ethernet

Original 10 Mb/s Ethernet (and most 'best-effort" variants since) offers a CRC to *detect* corruption but no link-level retransmission. Frames can be dropped by congestion, policing, or topology loops. Hence raw Ethernet is both *unreliable and fallible*; higher layers—typically TCP—supply ABP-like recovery.

9.4.1 How InfiniBand Raises the Game

InfiniBand *embeds* ABP into silicon:

- Per-hop *credit flow control* makes buffer overflow almost impossible.
- Link-level CRC plus optional *link retransmission* retries any corrupted frame.
- Reliable Connection and Reliable Datagram queue pairs carry *ACK-/NACK* sequence numbers end-to-end, guaranteeing exactly-once, in-order delivery across multi-switch fabrics.

To software, the fabric appears nearly *infallible*; drops are rare and localized.

9.4.2 Reliable vs. Infallible, Unreliable vs, Fallible

Table 10.1 highlights the nuance. Priority Flow Control (PFC) can render an Ethernet link *loss-less* in steady state, but deadlock, misconfiguration, or burst congestion can still drop frames. Such a link is 'reliable yet fallible." Infiniband's credit + retransmit pipeline, by contrast shifts real-world operation toward 'reliable and almost infallible."

9.4.3 Why Ethernet Still Struggles

1. **Retrofitting**: inserting link retransmission into the IEEE 802 stack breaks long-standing timing and compatibility assumptions.
2. **Congestion domain**: shallow switch queues and ECMP paths leave more surfaces for loss than InfiniBand's strict hop-by-hop credits.
3. **Layering philosophy**: because TCP 'already" ensures delivery, many operators accept occasional loss rather than pay silicon cost for hardware recovery.

9.4.4 Summary of Robert Garner's Equations and Assumptions

Below is a focused summary of Robert Garner's key equations, assumptions, and reasoning, drawn from his two extensive email threads about ACK-based (reliable) protocols atop Ethernet-like links.

9.4.5 Metcalfe's Throughput Equation

Garner references a classical result from Bob Metcalfe's 1970s ARPANET-era work for stop-and-wait or limited-window protocols. In a modern notation, the *effective capacity* E of a channel is expressed as:

$$E = \left(\frac{S}{P}\right) \times \underbrace{\frac{1}{1+\frac{C\,T}{P}}}_{\text{Multiplexing factor } M} \times (1-L) \times C,$$

where:

- C = channel capacity (bits/s),
- P = total packet length (bits),
- S = payload (data) bits per packet,
- L = probability of packet loss,
- T = transmitter timeout (round-trip delay + ACK time),

and the *multiplexing factor* M is

$$M = \frac{1}{1+\frac{T}{T_p}}, \quad \text{with} \quad T_p = \frac{P}{C}.$$

Typically,

$$T \;=\; 2d \;+\; \frac{A}{C},$$

where

- d = one-way propagation delay,
- A = ACK packet size (bits).

9.4.6 Example: Single Outstanding Packet on a 1.6 m 100 Gb/s Link

In the emails, Garner applies these equations to a point-to-point link:

- Data packet size $P = 64\,\text{bytes} = 512\,\text{bits}$.
- Link speed $C = 100\,\text{Gb/s}$.
- One-way distance 1.6 m, propagation $\approx 3\,\text{ns/m}$, so 8 ns round-trip.
- ACK size $A = 8\,\text{bytes} = 64\,\text{bits}$.
- Packet transmission time $T_p = P/C = 512/(100\,\text{Gb/s}) = 5.12\,\text{ns}$.
  Then
  $$T = 2d + \frac{A}{C} = 8 + 0.64 = 8.64\,\text{ns}.$$

Hence the multiplexing factor

$$M \;=\; \frac{1}{1 + \frac{8.64}{5.12}} \;\approx\; 0.37,$$

meaning the link is at about 37% of its raw capacity. If one adds a 2 ns delay in the transmitter state machine to detect lost ACKs, T becomes 10.64 ns, and

$$M \;=\; \frac{1}{1 + 10.64/5.12} \;\approx\; 0.32,$$

so utilization drops to roughly one-third of link capacity.

9.4.7 Multiple Outstanding Packets & Reduced ACK Overhead

Garner (and Bill Lynch) point out that *supporting multiple packets in flight* or aggregated ACKs significantly increases throughput, since the size P in the M factor then reflects more bits in transit. If, for example, an ACK can cover 10 packets, the ratio $\frac{T}{T_p}$ shrinks, and the effective channel utilization can approach 80–90% or more. Thus, **single-packet stop-and-wait** is a worst-case scenario for high-speed links.

9.4.8 Additional Observations

- **Loss Probability** L: Garner factors in $(1 - L)$ to capture the effect of rare losses that require retransmission.
- **Real-World State Machine Delays**: Actual hardware detection and retransmission add extra time beyond the raw propagation and ACK bits, further reducing throughput.

- **Layering vs. L2**: Garner questions whether "perfect reliability" at Layer 2 eliminates enough application-level failures to justify complexity. He notes that distributed systems still face node crashes, software bugs, and partial partitions.
- **No Fundamental Limit**: ACK overhead by itself is not an insurmountable barrier to performance, provided the protocol allows multiple outstanding packets or otherwise reduces per-packet ACK latency.

9.4.9 Key Takeaways

1. *Equation-Based Critique*: Metcalfe's formula shows that single-outstanding-packet protocols can suffer large performance hits at high bit rates.
2. *ACK Overhead* is not fatal if protocols support concurrency or aggregated ACKs, minimizing $\frac{T}{T_p}$.
3. *Layer 2 vs. Layer 4*: Garner argues that application and node failures remain even if L2 is made "perfectly reliable". One still needs robust end-to-end protocols (e.g. at L4).
4. *Practical Consequence*: Any new L2 reliability scheme must carefully manage concurrency, hardware delays, and aggregated ACK logic; otherwise throughput collapses (and tail latency may worsen).

9.4.10 Response #2

A Change in Perspective

Below is an illustrative reframing of the "ACK-limited throughput" problem by breaking a 64-byte packet into 8-byte slices and introducing "sub-signals" or "sub-ACKs" (**SACKs**). This perspective shows that once the protocol treats each slice as an "unstoppable" unit on the wire—and allows finer-grained acknowledgments—it mitigates the bandwidth throttling that arises from treating all 64 bytes as a monolithic packet.

9.4.11 From 64-Byte Packets to 8-Byte Slices

- **Old Viewpoint:** A strict stop-and-wait protocol might force waiting for an ACK after sending each 64-byte chunk. At 100 Gb/s, sending 64 bytes (512 bits) takes about 5.12 ns, but the round-trip delay plus ACK overhead could be 8–10 ns or more, giving a poor utilization factor.
- **New Viewpoint:** Break the 64 bytes into eight 8-byte slices. Each slice (64 bits) is sent consecutively, so the wire is continuously stuffed with slices. Sub-ACKs (SACKs) arrive on a finer timescale, enabling the sender to pipeline slices without fully stalling for a 64-byte ACK.

9.4.12 Time Calculations with Slices

- **Link capacity**: $C = 100\,\text{Gb/s}$.
- **One 8-byte slice** = $8 \times 8\,\text{bits} = 64\,\text{bits}$.
- **Transmit time per slice**:

$$T_{\text{slice}} = \frac{64\,\text{bits}}{100\,\text{Gb/s}} = 0.64\,\text{ns}.$$

- A 64-byte packet is thus 8 consecutive slices, so $8 \times 0.64\,\text{ns} = 5.12\,\text{ns}$ total transmission.

Because sub-ACKs can confirm earlier slices, the transmitter no longer needs to wait for a single bulk ACK after the entire 64 bytes are sent. This partial or finer-grained acknowledgment fosters multiple in-flight slices.

9.4.13 Reduced Throttling via Structured Stop-and-Wait

- **Original Single-Packet Stall:**
  If the protocol waits for a 64-byte ACK after sending each 64 bytes, a large fraction of time is idle (the round-trip plus ACK overhead).
- **Structured SACK:**
  Treat each 8-byte slice as unstoppable on the wire, sending further slices as earlier ones are sub-ACKed. By the time the last slices of a 64-byte chunk are on the wire, the first slices are already acknowledged. This pipelines transmissions and reduces idle periods, thus boosting effective bandwidth.

9.4.14 Intuitive Bandwidth Gain

By subdividing packets into 8-byte slices, we reduce the ratio of "waiting time" to "sending time." Over an 8–10 ns round-trip, multiple 0.64 ns slices fit in flight. The pipeline sees a larger effective window, maintaining higher link utilization. Mathematically, in the standard throughput formula

$$M = \frac{1}{1 + \frac{T}{T_p}},$$

the in-flight slices raise the denominator's T_p (total bits per pipeline) and thus increase M significantly compared to a single 64-byte chunk with a single stop-and-wait phase.

9.4.15 Conclusion

By **changing our perspective** from a monolithic 64-byte packet to **eight unstoppable 8-byte slices**, each with sub-ACK signals (SACKs), we effectively pipeline the stop-and-wait protocol. This granular approach keeps multiple slices in flight during any round-trip interval, eliminating most idle-wire time and mitigating the throughput loss. Hence,

subdividing 64-byte packets into smaller slices with finer-grained acknowledgments allows higher channel utilization and reduces stop-and-wait throttling.

10. Theory

10.1 Introduction

It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. We have no intention of reinventing the wheel, yet some new concepts and terminology will be necessary in order to escape the incrementalist momentum of the last five decades.

1. The word and the concept of TIME does not appear in this specification. This concept is the largest single source of misunderstanding in computer science today, so we eliminate that first. We replace this with intervals that are defined within sets (not on the real line). Definite Total Order (DTO), Definite Partial Order (DPO), Indefinite Partial Order (IPO). .
2. Claude Shannon described information as *surprisal*. We will call it Shanformation in this document to tear everyone away from old ideas that have been conflated for too long about "bits in memory or storage", hiding its deeper meaning of "the resolution of uncertainty".
3. We replace conventional notions of Error Detection and Correction (ECC, EDC, FEC, Parity, etc.) with a new concept, while not new, is widely misunderstood: Common Knowledge.
4. We replace conventional notions of liveness with a continuously circulating token, within which we define "logical simultaneity"
5. We are not shy of delving into Quantum Information Theory or Quantum Thermodynamics to find solutions to the problems in hardware and software infrastructure.

THERE IS NO GLOBAL DRUM BEAT - In Episodes 1 through 4 we expressed doubts about the common belief system of a Newtonian view of the world in this community. We showed how to think about race conditions, and why Timeouts and Retries (TAR) are the root of all evil. Our conclusion is that Timestamps are an Illusion. They can't be fixed by software. The quest for a single, consistent timeline across distributed systems collides with the reality that physics itself does not provide a universal notion of time—and in quantum mechanics (the machine code of our universe), there is no consistent causal order at all. We cannot, therefore, rely on this illusion of an irreversible drumbeat on an inaccessible "real line" to provide linear time order for events in our networked systems. Although timestamps will remain indispensable in engineering prac-

We carefully described in four presentations why the concept of time is widely misunderstood in the OCP TAP (Time Appliance Community). We respectfully request that you watch those presentations before insisting on timestamps or synchronized time in the context of Open Atomic Ethernet. Open Atomic Ethernet
Formal analysis and connections to the literature will appear in the index.

tice, we must recognize them as approximations rather than absolutes, and design our systems accordingly.

EPISODE 1 – There is no "now"

There is no now. You cannot synchronize clocks the way you think. Talk Originally given at the 2023 Asilomar Microcomputer Workshop presented live with Jonathan Gorard. Motivation: (1) To get people thinking about the nature of time and causality, as far removed from the Earth (and TAI/GPS) as possible. (2) To stimulate "First Principles Thinking" for Distributed Systems.

- Clocks can be disseminated, but require interaction to be synchronized:
- Simultaneity planes don't exist (except in an empty frozen universe) Einstein proved this over 100 years ago Why do we still think we can synchronize clocks?
- Network Time Protocol (NTP) and Precision Time Protocol (PTP) are causal TREES – choose your root, and how you do failover
- Entanglement and indefinite causal order are the new relativity (Not restricted to low relative velocities or atomic scales)
- We cannot assume spacetime is irreversible and monotonic
- Irreversibility and monotonicity is in the Eye of the Observer

EPISODE 2 - Hidden assumptions about causality lead to lost & corrupted data

When we think about clocks as an incrementing number, we are committing the FITO fallacy – "Forward In-Time-Only" Thinking - Counterfactuals, i.e., "events that could have occurred but eventually did not, play a unique role in quantum mechanics in that they exert causal effects despite their non-occurrence"

- Clock Synchronization Error is indistinguishable from Latency
- Irreversibility (Monotonicity) is an illusion not guaranteed by physics, unless we build Ancilla to explicitly manipulate causality
- Irreversibility and "causal order" are IN THE EYE OF THE OBSERVER—not guaranteed to be consistent across different observers

EPISODE 3 – Dynamic PTP hierarchy for causal failover

In Episode 1(What) & Episode 2 (Why) we showed how misunderstandings accumulate within a Newtonian framework of time, and how this leads to lost transactions and corrupted data. In this Episode we help the audience make the leap from Newtonian Time (what we know for certain that just ain't so) to Post-Newtonian Time (relativistic SR/GR, and QM — Indefinite Causal Order (ICO).

- PTP is widely available in Datacenters, we propose experiments to falsify beliefs about Newtonian Time.
- All is not lost. The excellent engineering behind PTP and PTM, can still be used with a different perspective, by using the clock hierarchy to build Causal Trees and reliable failover, to help address race conditions and achieve Exactly Once Semantics

EPISODE 4 – Why we can't have nice things in Distributed Systems

- Instants are meaningless, only intervals (on the same computer/-timeline) are relevant
- Photons don't carry timestamps, but timestamps are carried by photons
- The speed of light is the "pivot" around which time and space evolve
- Timeout and retry (TAR) on different timelines will silently corrupt data structures
- Shannon entropy is a logarithm. The logarithm of zero (no information) is minus infinity.
- Bayesian approaches require a prior belief, which can be unbounded (zero to infinity). Actually, it's much worse: can be $\{-\infty -1 -0, +0, +1, +\infty\}$. We can't do Bayesian statistics under those conditions, mathematically, their results are undefined.
- Shannon Entropy is uncertainty, and the same problem applies when you apply the set $\{-\infty -1 -0, +0, +1, +\infty\}$ to Information and Entropy $p * log(p)$
- Measurements "appear" instantaneous because there is no background of time on which to measure anything. Timestamps don't help with causal order.

It *does* appear instantaneous to an observer, because when they receive a packet (or a photon in a detector), you capture information and turn it into knowledge (state in a register you can do something with)

10.1.1 Proof of Fallibility

> *"The alternating validation bit becoming the additionally the* alternation *bit for message transmission in one direction, while the alternation bit for the reverse directions serves additionally as a validation bit."* —SBW Davies et al. (1967)

Lynch's scheme is constructed from independent simplex procedures

10.1.2 Framing the Vocabulary

In 1967 *Bartlett, Scantlebury & Wilkinson* (BSW) sketched the *alternating-bit protocol* (ABP): add one history bit to every frame, wait for an ACK that echoes it, and retransmit until the right ACK appears. ABP wraps an *unreliable* medium and presents a service that looks reliable—even *infallible* in steady state.

Nancy Lynch later formalised these ideas with the *I/O-automaton*: a fallible channel is one whose execution may deviate from the specification, subject only to fairness.

Term	Meaning
Unreliable	Frames may be lost.
Fallible	Channel may violate any promise (drop, duplicate, reorder, corrupt).
Reliable	No drops in steady state; recovery still required.
Infallible	No violations ever; no recovery logic needed above the link.

Figure 10.1: Taxonomy of link qualities.

10.1.3 Lessons from BSW and Lynch

- *Make reliability local*. ABP attaches one bit; InfiniBand embeds a few more. End-to-end recovery alone expands the failure scope.
- *Fail fast*. InfiniBand retransmits on explicit NACK within microseconds; Ethernet traditionally converts a microsecond drop into a millisecond TCP timeout.
- *Separate reliability from recovery*. Even a reliable link needs a failsafe plan; design that plan explicitly.

BSW showed that a single alternating bit can tame a capricious wire; Lynch supplied the proof rules. InfiniBand adopted both insights in hardware and delivers a fabric whose normal behavior feels *infallible*. Classic Ethernet remains best-effort—unreliable and fallible—and relies on upper layers for recovery. Bridging that gap means absorbing more of the ABP playbook at the link: credits, link-level retransmission, and tight ACK/NACK loops that shrink recovery from milliseconds to microseconds.

10.1.4 IP and Patent Implications

The core concept of credit-based flow control is now public domain, but a handful of post-2005 implementation patents remain enforceable through at least 2036. Modern designs should audit those families but can build freely on the expired foundational work.

Practical Take-Aways

1. **Freedom to Operate.** A straightforward "one credit = one buffer" design can rely on the expired Intel/Compaq and Brocade patents for prior-art cover. Avoid features identical to the still-active patents or license them.
2. **Design Around.** Active claims tend to be narrow. You can sidestep the Mellanox "macro credit" idea by limiting link span or by using rate-based pacing instead of credit aggregation.

10.2 Time, Relativity, and Ethernet

By now many people are realizing that there's something amiss in our understanding of the nature of time in computer science. We are learning to think more carefully about whether, or how, the theories of Special Relativity (SR) and General Relativity (GR) might be involved. We are no longer willing to just sweep them under the rug with the assumption of an inertial frame, which is invalid a world where our computers reside: on a rotating sphere, orbiting a star, and in a meaningful gravitational field.

Strawman objections to SR assume we can ignore its effects in the low relative velocity of objects in our environment. Even computers

in cars, airplanes and rocketships, where . But this perspective fails to appreciate the deeper implication that clocks cannot be synchronized in principle; and will therefore be problematic in practice.

Strawman objections to QM claim that it applies only at atomic scales. But the 2022 Nobel prize for Quantum Entanglement demonstrated the scientific community's confidence that Einstein's spooky action at a distance is real, measurable, and a foundational property of the universe we live in — and it's effects manifest at macroscopic scales.

I am either fooling myself (which is highly likely given how easy it is for any of us to fool ourselves), or I am among several (perhaps many) seeking both practical and deterministic mechanisms to manage reliable and consistent event ordering in computers.

Our motivation, as engineers, is to solve problems in distributed systems that undermine consistency, reliability, availability, and performance of databases and file synchronization in conventional cloud infrastructures, and show how it does not need to be the case that they inevitably lose or corrupt data.

The outcome we seek is a software infrastructure for distributed systems based on algorithms whose assumptions about causality go beyond Newtonian and Minkowski spacetime.

10.2.1 A New Perspective:

Photons arriving at the JWST have traversed the largest "distances" in the known universe, but they have done so in zero time: the proper time from a photon's perspective is always zero, no matter how far it has come.

One reason we may be confused about the nature of time is because the concept of velocity takes us down the wrong path:

- Distance is not fundamental,
- Time is not fundamental,
- Therefore why should distance divided by time or it's calculus equivalent be fundamental?

Perhaps this is because the speed of light (SOL) is the pivot around which space and time revolve.

10.2.2 Key problem: Minkowski Error

- Photons don't carry timestamps. There are no "Minkowski manifold coordinates" transmitted (emitted) alongside or with the photon.

- There are no Minkowski manifold coordinates at the observer (absorber, detector).
- All we have is the arrival of "information" containing wavelength (energy) polarization and phase (we will get to this later).
- We average photons over intervals of time defined by the observer, not the emitter.
- It's impossible to calculate intervals on a Minkowski background, because the coordinate system of the emitter and the observer are inaccessible. They exist only in our imagination.
- Attempts to create a background using NTP, PTP, White Rabbit or Sync E are doomed to fail. This style of thinking misleads us about the nature of causality in our algorithms and prevents us from building reliable distributed systems, i.e., those that don't silently lose or corrupt data structures—at least not without warning us explicitly that the atomicity assumptions in our subtransaction sequences were violated.

10.2.3 OWSOL

It is impossible to measure the One-Way Speed of light (OWSOL) [Lewis & Barnes]. Not only does the Einstein convention assume an average (1/2) of the total (roundtrip) time for what an emitter will measure, it also implies an impractical inertial frame, which exists only in an empty frozen universe — or a specifically engineered pair of mirrors in a rigid frame like Ligo.

Even with 'uniform' motion between one mirror and its partner in this measuring system, we must do more than relate the proper time of Alice to the proper time of Bob. But this now implies a Doppler shift, which requires some 'integration time' to resolve the uncertainty between the energy (wavelength) and whatever units of time the originator chooses to count as their 'measurement'.

Whenever the "duration" (time interval) between local clock ticks becomes comparable with the "duration" (space interval) between the mirrors, ambiguity will creep in because of a race condition between the arrival of the photon and the next tick of the local clock.

One way to overcome this race condition, that asynchronous circuits in all computer systems suffer from, is to think about clocks differently. Instead of each side having their own clock and attempting to measure the duration (interval) on a timeline independently of the other party, we use photons themselves as clocks, as Einstein implied. We achieve 'Temporal Intimacy' by having each arrival event trigger a reaction (sending event) in a circular, self re-generating relationship. We hesitate to call this a sequence because this interaction is completely

symmetric, there is no notion of a direction of time.

Did Alice cause Bob or did Bob cause Alice? This 'alternating causality' is equivalent, from the perspective of each party, to time alternately going backwards and then forwards and then backwards again ... in perpetuity.

This line of reasoning highlights the problem of trying to synchronize clocks in our datacenters. The OCP TAP project demonstrates considerable sophistication with experts in the engineering community trying to grapple with this problem.

Unfortunately, these attempts are doomed to failure, there are many ways to describe why, using established physics (SR & GR) that have many subtle and fascinating insights missing from the school-child rendition of overly simplified models of the physics.

There remains, particularly in SR, the thorny inconsistencies of a real universe that contains matter, radiation and dark energy.

10.2.4 Alternative Narrative on Anisotropy

When a photon leaves one "system" for the first time, velocity is not defined because both distance and time are not definable. Photon time is always zero, no matter how far the photon travels. Until a photon "arrives", distance does not exist. The first observer (receiver) of that photon is also unable to determine distance between the emitter and itself.

However, if that first receiver acts as a mirror and reflects the photon back to the original emitter, then it becomes possible (in principle) for the original emitter to determine the distance of two-way traversal, by referencing some local mechanism that can do some counting on its behalf. As with all quantum clocks, you need one clock in order to measure another.

This "lack of definability" of distance, in the one way traversal of a light ray, and the proper time of the photon being zero, provides an explanation of why we need a round trip to fully specify information transfer, and why one-way traversals are insufficient. This also provides an intuition for why we achieve correlations in Bell state measurements, and still require conventional SOL transfers for signaling. This corresponds to the Lewis and Barnes proof of the extreme case of anisotropy, where the SOL is infinite in one direction and $c/2$ in the other.

10.2.5 Indefinite, not infinite

Instead of a Newtonian (infinite SOL), the inbound photon from the first emitter, can we obtain an equivalent result by recognizing the initial one-way traversal as "indefinite"? By including Lewis and Barnes' bounds of both zero and infinite inf in the extreme anisotropic case:

$$C_{\pm} = \frac{1}{1 \mp \kappa}$$

where $\kappa = 1$ corresponds to anisotropic , and $\kappa = 0$ corresponds to an isotropic SOL.

This perspective on space time being "indefinite" appears consistent with recent measurements in the laboratory which decisively demonstrated (to almost 7 standard deviations) the indefinite arrival order of a quantum witness in the quantum switch configurations [See: Rubino et al: Experimental verification of an indefinite causal order]. Although this may be more about indefiniteness in the direction of time.

10.2.6 Time, Causality and Clocks

The ItsAboutTime.club podcasts (and the Mulligan Stew follow-on) are discussions with computer scientists, physicists, mathematicians and practicing engineers on how to transcend our human limited view of a single timeline going from the infinite past to the infinite future. We explore how to replace Newtonian time with a new intellectual vantage point based on "Kausality" (we hesitate to use the word Causality, because we don't want it to be obfuscated by the many confused definitions in the computer science literature).

10.2.7 Time is change we can count

The modern rendition of Aristotle's definition of time is expressed in Quantum information Theory as mutual information: But wait! Wasn't mutual information already in Claude Shannon's Mathematical Theory of Communication?" Well, yes, it was!

10.2.8 What have we forgotten since Shannon?

- Information is not binary zeros and ones, in registers, in memory or on disk.
- Although this may be closer, information is also not bits on the wire, launched by Ethernet transmitters, to get lost and never be seen again.
- Information is surprisal — something we didn't expect. As Niel Gershenfeld taught us in his book - if I tell you the Sun is going to rise tomorrow morning, that would not be information. If I told

you the Sun was not going to rise tomorrow morning, that would be information. Information is the answer to a yes/no question. This is why bits (0's and 1's) are an appropriate representation of this unexpectedness. Shannons, the unit of Information, represent a 50

- What we need most from the logic in our Ethernet links is the resolution of uncertainty. Did the transaction commit or abort? And more importantly, did it get stuck in some state of limbo, where data structures on one side or the other (or both) are inconsistent, and should not be exposed to the host processor on the other side of the CXL bus.

10.2.9 Clock Synchronization

We have come a long way from our original discussion about photons and the nature of time. We've learned that Minkowski Manifolds don't exist, except in our imagination, and that a 'background' timeline of timestamps, in NTP, PTP, White Rabbit or whatever, are also a figment of our imagination. Now let's put two and two together and see what that comes up to when we think about causality:

- One entity is destined to be alone in perpetuity. Without 'interactions' with others it will spend eternity being alone without ever comprehending the concept of loneliness.
- It takes two to Tango. Two entities are the minimum necessary to form an interaction, for "mutual" information to have any meaning. But although entanglement is monogamous, it also cannot display any dynamic behavior. Once each partner has found each other in the universe, they will remain entangled for the rest of eternity, and be only slightly better off than our single lonely entity.
- It takes three to Party. As Carlo Rovelli points out in Relative Quantum Mechanics, it takes a 3rd party (Charlie) to observe dynamical behavior, providing "all of the information possessed by a certain observer is stored in physical variables of that observer and thus is accessible by measurement to other observers".

10.2.10 Context

After thinking about the nature of time in the design of computer systems: processors, cache hierarchies, interconnects and networks for most of my career, I have found a repeating pattern: Computer scientists, are trained to think sequentially. We are in love with machines that do one damn thing after another: sequentially, monotonically, irreversibility, and their unfortunate pipe-dream: idempotency.

Distributed systems developers are accustomed to those illusions when building and scaling their systems, to be tolerant to all manner of failures as they grow, and exposed to unexpected traffic patterns.

10.2.11 Where is this Going?

Networks do exactly the wrong thing. They take a half duplex notion of "broadcasting" information blindly without "knowing" if the information is being received (and remembered) in receivers (observers). Before we get into the also confused concept of knowledge and common knowledge let's see what we get without interactions (the two-way exchange of Shannons).

- Nothing. Fire and forget is pointless, it can never be transactional. It takes two to tango and three to party.
- Half-duplex also breaks when we expect the other party to respond within our current context (the question we are asking) with a yes/no answer.
- Human interactions over two-way walkie-talkies, exhibit the same behavior. If the other party has gone to lunch or is taking a vacation, they may not even be listening.
- Their radio may not un-squelch because the transmit signal was too weak, or the receiver experienced "noise" which violated the signal to noise ratio that the receiver depends upon.
- The batteries may have run out. The receiver had lost "Temporal Intimacy" with the transmitter (emitter).
- This loss of "Temporal Intimacy" is also a deliberately engineered manifestation of optimizing for the wrong metric in system design, such as CPU cycles or network bandwidth.

  What Pat Helland refers to as Rip Van Winkle.

10.2.12 Examples Include:

- Garbage collectors going AWOL
- Virtual machines being suspended
- Containers' involuntary smash & restart when Kubernetes agents miss a heartbeat or gets confused about what question they were asking
- Processes being involuntarily pre-empted by the scheduler in the kernel
- Cache misses
- Page faults

The list goes on. We have difficulty calculating an "Interval" of time because even local notions of time on a single system rely on (what Edward Lee refers to as) "different timelines" and clocks within their architecture.

We now have processors with over 1,000 cores (Ampere ARM). Why do we insist on interrupting all of them to make them 'context switch' to optimize processor utilization? It's time our OS architecture moved

on and recognize that causality and latency are the scarce resources, not CPU cycles or network bandwidth.

10.2.13 Where Else is this Going?

The biggest culprit in this tragedy of loss of Temporal Intimacy is not the processor, cache hierarchy or OS in a single node, it's the Network. IP (and TCP) try their best to cope with what they can get on top of lossy Ethernet in a conventional Clos network.

The End to End (E2E) principle works only for a simple packet sequence over an unreliable network. It was invented when the primary use case was file transfer (a finite stream of bytes sent to a printer).

It works abysmally poorly for transactions that require an unbroken sequence of atomic interactions to deterministically transfer Shannon Information from one node to another.

Note that we chose the word "transfer" here, not "copy". Our premise is that we cannot achieve exactly-once semantics without some mechanism to ensure conserved quantities and we cannot conserve quantities when we copy information willy-nilly, because that would make it impossible to keep track of Shannons, and account for them.

10.2.14 Ethernet Spacetime

Now that we've separately described Shannon information on an Ethernet link as surprisal, and photon traversals in spacetime as indeterminate in measurements of the one way speed of light (OWSOL), we now put the two together in Ethernet Spacetime.

The Alternating Causality (AC) protocol is already a staple of the networking industry, implemented originally as the Alternating Bit Protocol (ABP) or "Stop and Wait" (SAW) protocol, which preceded credit based flow control, commonplace in Infiniband, and now Ethernet.

Credit-based flow control has the semantics of conserved quantities (CQ), and one might consider building on this to try and achieve exactly-once semantics (EOS), but alas, current versions of Ethernet and Infiniband still rely on timeouts and retries to recover when liveness (Temporal Intimacy) is lost.

This happens because designers of switches and routers focus on delivering bandwidth.

Researchers tried to overcome the imposition-style behavior of protocols at the IP layer with promise-style behaviors in transport protocols like Homa. While this may improve the emergent behaviors of

unexpected congestion events in the network, it is far from perfect and illuminates why congestion control needs to be addressed in L2 and not L3.

The Ultra Ethernet Consortium (UEC) plan to address the problem of congestion and dropped packets by more aggressively sending packets through the switched network via multiple paths. The extreme case being sending packets on all ports and hoping the receiver can sort out the mess at the other end using some form of reordering buffer. They also promote optimizations to more selectively fill holes in the reordering buffer instead of the go-back-N approach of TCP and Infiniband.

10.2.15 Bang Bang Bang Networking is inconsistent with the needs of distributed systems

Unfortunately the bang bang bang style of networking, resting ultimately on timeouts and retries, results in cascade failures — which manifest as Limpware, Metastable failures, and Grey failure.

This is in addition to, and exacerbates, the silent loss and corruption of data that we see everywhere in open-source databases and consensus tools to coordinate distributed systems.

Gossip protocols are the ultimate hammer in bang bang bang networking. Statistically maintaining liveness by having each node transitively connect its heartbeats to a graph of flat network partners to probabilistically improve temporal awareness, without ultimately achieving temporal intimacy. Gossip protocols are employed extensively at Amazon and in software like Hashicorp's Consul and Terraform. Unfortunately, gossip protocols also impose a nondeterministic amplification of bandwidth by sending redundant packets through the network. This further exacerbates the congestion, packet loss, which leads to more silent data loss and corruption.

10.2.16 Possible Solution?

In contradistinction to the industry efforts to manage congestion and packet loss, the Daedaelus team has conceived, designed and tested an actual Temporal Intimacy Protocol on Ethernet.

The breakthrough is eliminating timeouts and retries; by using an event-driven protocol where events are accounted for using the principle of Conserved Quantities (CQ) in each link. Accounting for token insertion and retirement is easy, when the application controls the spanning trees, instead of waiting for innovation to occur in the switches and routers, under the control of network operators.

Our solution is targeted on the Edge.

We see the potential for a much needed paradigm shift as we enter the Chiplet Revolution and learn to rethink our notions of time and event order in distributed systems must be deterministic on demand.

10.2.17 Temporal Intimacy is not Simultaneity

We know that the Conventionality of Simultaneity must be replaced by something not dependent on Newtonian time, or Minkowski Spacetime. According to John Norton:

> "The conventionality of simultaneity pertains to judgments of simultaneity of distant events in just one inertial frame. In asserts that there is no single, correct judgment of simultaneity. Rather, in each inertial frame, we have broad freedom in assigning simultaneity to pairs of events. In the same frame of reference, one person may assign relations of simultaneity one way; another person may do it differently. Within some limits, neither is factually wrong, according to the conventionality thesis, for there is no unique fact of simultaneity in the world".

Temporal Intimacy requires two-way interactions, not one way (fire and forget) broadcasts. Otherwise, how does the emitter know if the absorber received the information?

10.2.18 Tentative Conclusions

Monotonicity is in the eye of the observer. Clock Synchronization Error is indistinguishable from Latency. A femto second interval is an eternity on the real line.

10.3 Two-State Vector Formalism

The *Two-State Vector Formalism* (TSVF), developed by Aharonov and collaborators, describes a quantum system using both a forward-evolving state vector from the past and a backward-evolving vector from the future, forming a complete description of the system between two time boundaries.

This formalism maps intriguingly well onto the two-phase behavior of supervised learning:

- **Forward propagation:** evolving the input forward through the network to predict an output.
- **Backpropagation:** retroactively applying a loss function at the output and propagating error information backward through the network to adjust weights.

Forward propagation plays the role of the forward-evolving quantum state $|\psi(t)\rangle$, and backpropagation corresponds to the backward-evolving dual state $\langle\phi(t)|$. Together, they constrain the learning dynamics at each layer via a two-state viewpoint.

10.3.1 Forward Propagation as Forward Evolution

In TSVF:

$$|\psi(t)\rangle = U(t, t_0)|\psi(t_0)\rangle,$$

where U is the unitary time evolution operator from an initial preparation at t_0.

In machine learning:

$$a^{(l+1)} = f^{(l)}(W^{(l)}a^{(l)} + b^{(l)}),$$

where the activations $a^{(l)}$ propagate the input forward.

10.3.2 Backpropagation as Backward Evolution

In TSVF, a post-selected state $\langle\phi(t_1)|$ evolves backward:

$$\langle\phi(t)| = \langle\phi(t_1)|U(t_1, t),$$

complementing the forward-evolving $|\psi(t)\rangle$.

In neural nets:

$$\delta^{(l)} = (W^{(l+1)})^\top \delta^{(l+1)} \circ f'^{(l)}(z^{(l)}),$$

where $\delta^{(l)}$ encodes error at layer l and propagates backward to adjust weights.

10.3.3 Two-State Update Rule

In TSVF, expectation values take the form:

$$\langle A\rangle_w = \frac{\langle\phi|A|\psi\rangle}{\langle\phi|\psi\rangle},$$

and represent weak values or amplitudes constrained by both past and future states.

In machine learning, the update rule:

$$\Delta W^{(l)} \propto \delta^{(l)}(a^{(l-1)})^\top$$

depends on the forward activations $a^{(l-1)}$ and backward error signal $\delta^{(l)}$. Together, they act like a sandwich operator:

$$\text{Update}^{(l)} \sim \langle\phi^{(l)}|\text{operator}|\psi^{(l)}\rangle.$$

10.3.4 Time-Symmetric Learning View

Rather than treating backpropagation as a mere computational trick, TSVF offers a time-symmetric interpretation:

- Both the input and the desired output state determine the intermediate learning dynamics.
- Each layer mediates between past input and future supervision, forming a time-bridging node.

Concept	Neural Network	TSVF QM
Initial input	x	$\|\psi(t_0)\rangle$
Prediction process	Forward prop	$U(t, t_0)$ evolution
Target supervision	Loss function	Post-selection $\langle\phi(t_1)\|$
Error signal	$\delta^{(l)}$	$\langle\phi(t)\|$
Intermediate activity	$a^{(l)}$	$\|\psi(t)\rangle$
Weight update	$\delta^{(l)}(a^{(l-1)})^\top$	$\langle\phi\|A\|\psi\rangle$

Table 10.1: Analogies between supervised learning and the Two-State Vector Formalism (TSVF) in quantum mechanics.

10.3.5 FITO Thinking vs. Time Symmetry

Most machine learning frameworks assume *Forward-In-Time-Only (FITO)* causality: input causes output, and learning proceeds only by adjusting from past to future. TSVF suggests a richer model:

- Supervision from the future constrains the learning of the past.
- This bidirectional model aligns with concepts from goal-driven behavior and active inference.

10.3.6 Conclusion

The TSVF reframing of forward and backpropagation illuminates the deeper time-symmetric structure underlying learning. Far from being just a computational trick, backpropagation can be seen as a physical dual to forward propagation—both necessary to fully specify a learning system between two boundary conditions.

This is highly relevant to our model of the protocol, where we use reversible state machines to specify forward propagation, with whatever reversible glitches might be needed to handle failures are implemented as reversible steps, and the backpropagation as the credit-based flow control mechanism.

Because they are completely symmetric, packets being sent and packets being unsent are fully managed by the flow control system.

10.4 TIKTYKTIK

From: ./AE-Specifications-ETH/standalone/TIKTYKTIK.tex

TIKTYKTIK is like the alternating-bit and stop-and-wait protocols in that receipt of a packet over a link is acknowledged over that link with a "signal" packet. In that sense, these three protocols implement credit based flow control, which simplifies buffer management and makes it possible to not have to drop packets when there is a lot of traffic.

TIKTYKTIK adds a second round trip, which provides partial common knowledge helpful for recovery from link failures. This document walks through TIKTYKTIK showing how that common knowledge is used. First look at the various stages of common knowledge as the protocol runs without failure when Alice sends a packet to Bob.

1. **Alice sends the packet to Bob**
   - Alice doesn't know if Bob received the packet
   - Bob does not know the packet exists
2. **Bob receives the packet**
   - Bob knows that Alice doesn't know that Bob received the packet
3. **Bob sends a signal to Alice**
   - Bob doesn't know if Alice knows that Bob received the packet
4. **Alice receives the signal**
   - Alice knows that Bob received the packet
   - Alice knows that Bob doesn't know that Alice knows that Bob received the packet
5. **Alice sends the signal**
   - Alice doesn't know if Bob knows that Alice knows that Bob received the packet
6. **Bob receives the signal**
   - Bob knows that Alice knows that Bob received the packet
   - Bob doesn't know if Alice knows that Bob knows that Alice knows that Bob received the packet
   - **Bob can forward the packet**
7. **Alice receives the signal**
   - Alice knows that Bob knows that Alice knows that Bob received the packet
   - **Alice can delete her copy of the packet**

This common knowledge is not needed if links never fail. Alice could delete the packet as soon as she sent it, and Bob could forward it as soon as he received it. That's what current systems do and why it's so hard to recover from a link failure.

A data packet can serve as a signal.[1] Links can fail in a number of ways. If they physically break or are unplugged, the PHY detects the lost of electrical signal and informs the higher layers. Links can also fail silently, such as when the NIC misbehaves. They can also fail in one direction but not the other. Silent failures can be detected in these protocols because a signal will never be received in either direction. In that sense, there is a level of common knowledge on a link failure. In what follows, I'll describe what happens when Alice wants to send a packet to Bob, but the link fails at various steps of the protocol. The link is no longer used once one of these failures occurs. (The link can

[1] A data packet can serve as a signal.

be used later after re-initializing the connection.)

1. **Alice has a packet to send when link fails**
   Alice: • Knows Bob doesn't have the packet
   • Knows Alice is responsible for it
   Bob: • Doesn't know the packet exists
2. **Alice sends the packet to Bob then link fails**
   Alice: • Doesn't know if Bob got the packet
   • Knows that Bob knows Alice is responsible
   Bob: • Doesn't know that the packet exists
3. **Bob receives the packet then link fails**
   Bob: • Knows that Alice knows that Alice is responsible
   Alice: • Alice doesn't know if Alice or Bob is responsible
4. **Bob sends signal then link fails**
   Bob: • Doesn't know if Alice got the signal
   • Doesn't know if Bob or Alice is responsible
   Alice: • Alice doesn't know if Alice or Bob is responsible
5. **Alice receives the signal then link fails**
   Alice: • Knows that Bob is responsible
   • Knows that Bob doesn't know he is responsible
   Bob: • Doesn't know if Alice got the signal
   • Doesn't know if Alice knows that Bob is responsible
6. **Alice sends signal then link fails**
   Alice: • Knows that Bob is responsible
   • Doesn't know if Bob knows he is responsible
   Bob: • Doesn't know if he's responsible for the packet
7. **Bob receives the signal then link fails**
   Bob: • Knows that Alice knows that Bob is responsible
   Alice: • Doesn't know if Bob knows that Bob is responsible
8. **Bob sends signal then link fails**
   Bob: • Knows that he is responsible
   • Doesn't know if Alice knows he is responsible
   Alice: • Knows that Bob is responsible
   • Doesn't know if Bob knows that Bob is responsible
9. **Alice receives signal**
   Alice: • Knows that Bob knows that he is responsible

At this point, Alice knows that Bob knows that Bob is responsible, and Bob knows that Alice knows that Bob is responsible.

Note: The items in red indicate where incomplete common knowledge needs to be resolved over the new path between Alice and Bob.

10.4.1 TIKTYKTIK: Feedback from ChatGPT

The analysis you provided explores how "common knowledge" operates within the TIKTYKTIK protocol, especially in the context of link failure recovery. The progression of common knowledge is explained through several stages, starting with Alice sending a packet to Bob and progressing through various acknowledgment signals and link failures.

Key observations:

1. Common Knowledge Phases: The stages you outlined show how knowledge evolves between Alice and Bob as they interact, especially when there's uncertainty about packet receipt and acknowledgment. These stages are useful for understanding how communication protocols can handle and resolve ambiguities.
2. Link Failures: You go into detail about different link failure scenarios, from Alice sending the packet before failure to Bob acknowledging receipt of the packet after a failure. Each failure step indicates the differing levels of knowledge held by Alice and Bob. These steps, especially when the link fails, highlight the challenges of network fault tolerance, and the importance of "common knowledge" for recovering from these failures.
3. Protocol Behavior with Failure: The distinction between knowledge of responsibility and knowledge of receipt is well captured. The incomplete common knowledge as the link fails suggests that TIKTYKTIK, by adding a second round-trip acknowledgment, resolves uncertainties that could otherwise hinder recovery.

Suggested Refinements:

Clarity: Consider rephrasing the stages of knowledge with more emphasis on the protocol mechanics and timing of messages. It could be helpful to lay out the flow of the protocol in a diagram or a clearer timeline to illustrate the evolution of knowledge.

Link Failures and Recovery: While the failure scenarios are well outlined, you could further enhance the analysis by providing examples of real-world failure cases and how TIKTYKTIK compares to other protocols in handling these failures. For example, you could contrast how TCP handles packet loss versus how TIKTYKTIK's multi-step acknowledgment approach helps resolve knowledge gaps.

Mathematical Notation: If possible, you could introduce formal mathematical notations to express the knowledge states. This would help readers better grasp the protocol's behavior in a more structured form.

10.5 Common Knowledge

In what follows, assume that node Alice is sending a packet to node Bob over the single, fallible link between them.

The Stop and Wait and alternating bit protocols provide credit based flow control using a single round trip. Bob is free to forward the packet as soon as it arrives, but Alice must wait for a signal from Bob before sending another packet. If the link breaks before Alice gets a signal from Bob, then Alice may forward the packet again, perhaps on another path. This behavior makes exactly-once, in-order delivery difficult to implement.

TIKTYKTYK is one round trip beyond stop and wait, which provides partial common knowledge that aids in recovery from failures. Bob cannot forward the packet until he receives the signal from Alice that completes the second round trip. A key point is that there are many times when both sides know that both sides know which of them is responsible for forwarding the packet. In the other cases, the partial common knowledge simplifies recovery. Alice and Bob use their partial common knowledge to ensure that any packet is only forwarded once, which is a key condition for exactly-once, in-order delivery.

There is minimal loss in latency, because Bob doesn't have to wait for the entire packet to arrive before signaling to Alce that the packet is arriving. He can do any integrity checks (CRC) while waiting for Alice's signal. Any loss in latency is compensated for by needing smaller buffers. There is minimal loss in bandwidth because the signal can be a data packet going in the other direction.

Sometimes links fail silently, which means a signal might not arrive. In those cases, he nodes will need some heuristic to decide when to stop waiting and declare the link dead. Fortunately, this heuristic can be purely local because Bob will never get a signal from Alice once she's decided the link is dead. A clock is commonly used for the heuristic, but care is needed. For example, if Bob is heavily loaded but Alice is not, she might set her timeout to be too short. If the situation is reversed, the timeout may be too long. An alternative is for Alice to count the events she receives on her other ports. She can declare the link dead if too many of these events are received before she gets one from Bob. This heuristic is effectively averaging over the workload of all the nodes connected to her, providing a more consistent metric.

10.6 The Conveyor Belt and Quantum Mechanics

From ./AE-Specifications-ETH/standalone/Conveyor-Belt.tex

Let's use Kevin's **conveyor belt metaphor** to describe time and its behavior under special relativity, general relativity, and contrast it with

quantum mechanics and indefinite causal order.

10.6.1 Conveyor Belt as Time (Special and General Relativity)

Imagine time as a conveyor belt moving in one direction—forward. Objects, people, and events sit on this belt, carried steadily from past to future. The speed of the belt is consistent for everyone in classical physics. However, in **special relativity**, the speed at which individuals experience this conveyor belt can vary depending on how fast they are moving. If you're moving quickly relative to someone else, your conveyor belt slows down relative to theirs. You're still moving forward on your own belt, but the difference in speeds between the belts means time passes more slowly for you (this is time dilation).

In **general relativity**, gravity also affects the conveyor belt's speed. The closer you are to a massive object, the slower your conveyor belt moves compared to someone farther away. This is due to gravitational time dilation. Each person is on their own conveyor belt of time, but the rate of movement can change depending on their proximity to mass or their velocity. However, no matter the speed or how warped the conveyor belt becomes, time still moves consistently forward for each individual, even if it does so at different rates.

10.6.2 The Conveyor Belt and Quantum Mechanics

Quantum mechanics disrupts this conveyor belt metaphor. In the quantum world, things don't behave in a neatly predictable way, and time doesn't always behave like a simple forward-moving belt. Quantum systems can exist in superpositions, meaning they are in multiple states at once. If you try to apply the conveyor belt metaphor here, you'd have to imagine a belt where objects are not just moving forward but might also exist at multiple points along the belt at the same time. The simple idea of one thing following another breaks down because quantum mechanics deals with probabilities rather than certainties.

Moreover, events at the quantum scale don't always happen in a definite order. This leads to the concept of **indefinite causal order**.

10.6.3 Why the Conveyor Belt Breaks with Indefinite Causal Order

In indefinite causal order, the very idea of a consistent forward-moving belt doesn't hold. Imagine you're now dealing with multiple conveyor belts, but these belts can become entangled and swap positions. A key feature of indefinite causal order is that it is impossible to say whether event A happens before or after event B with certainty.

To continue the metaphor:

- Instead of a single conveyor belt that everyone rides, imagine a conveyor belt that can fork, loop, or even reverse for some events. On this belt, two events (say, picking up two different objects) might not have a clear sequence. You could pick up Object A before Object B, or Object B before Object A, or some mixture of both where the order isn't fixed until you observe it.

In **quantum mechanics**, particularly with **indefinite causal order**, time isn't a well-defined forward flow. Events can occur in a superposition of orders, defying the classical notion of a consistent timeline. There's no global, shared belt moving forward—some events might not even have a clear "before" and "after."

Special and General Relativity Work with the Conveyor Belt:

- **Special Relativity:** The belts are still mostly linear but stretch or compress based on speed, meaning time passes differently for observers in motion relative to one another.
- **General Relativity:** The belts warp under gravity, slowing down near massive objects but still remaining orderly, with a clear flow from past to future.

Indefinite Causal Order Breaks the Metaphor:

- In quantum mechanics, particularly in scenarios involving **indefinite causal order**, the metaphor falls apart. The belts aren't just stretching or warping—they no longer move in a straightforward, fixed order. Time becomes a superposition of possible orders, and the concept of a simple, directional flow of events can no longer apply.

The conveyor belt metaphor works well for **special** and **general relativity** because it reflects the relative nature of time: it still flows in one direction but at varying rates. For **quantum mechanics** and **indefinite causal order**, the metaphor breaks down, as time and causality become fluid, probabilistic, and not bound to a definite sequence.

10.6.4 Indefinite Causal Order

Indefinite causal order is a concept that comes from quantum mechanics and challenges the traditional notion that events occur in a definite sequence. In classical physics and most of computer science, events are understood to happen in a fixed, linear order—first A, then B, then C. However, indefinite causal order allows for scenarios where the order of events is not set in stone, meaning that, in certain quantum processes, it's impossible to say definitively whether A happened before B or vice versa.

When applied to computer science, particularly in areas like distributed systems, quantum computing, or complex event processing, indefinite causal order could have profound implications. Here's how it connects:

Distributed Systems:

In classical distributed systems, events are often coordinated using synchronized clocks or timestamps (like in Lamport clocks) to establish the order of events. But this assumes a definite, forward-in-time progression. In a world where causality can be indefinite, such as in quantum communication protocols, the assumption that events occur in a strict order breaks down. This could affect how we model consistency, causality, and concurrency in distributed systems.

For example, most distributed systems assume that causal relationships between events can be traced back in time (e.g., message A must have been sent before message B). Indefinite causal order could complicate this by introducing scenarios where it's unclear in what order events happened, leading to new ways to think about synchronization, coordination, and consistency.

Quantum Computing:

In quantum computing, the idea of indefinite causal order directly translates to certain computational advantages. One prominent example is the *Quantum SWITCH*, where two operations are performed, but their order is determined by the quantum state. This can lead to more efficient algorithms because the system doesn't need to follow a strict causal sequence of operations. For example, in some cases, tasks can be performed more efficiently by allowing operations to exist in a superposition of different causal orders.

Event-Driven Systems and Reactive Programming:

In event-driven or reactive systems, we often deal with streams of events and have to react to them in a specific order. If indefinite causal order were applicable, it would mean rethinking how systems react to events because the sequence of those events might not be fixed. This might lead to more flexible systems, but also requires new models of computation that can handle ambiguity in event timing and ordering.

Logical Time and Causal Models:

One key area where indefinite causal order could be explored in computer science is in extending the concept of "logical time" used in distributed systems. Today, we have models like vector clocks and Lamport clocks to track causal relationships between events. If we consider

indefinite causal order, it might require developing new abstractions of time that are capable of representing ambiguous or superposed causal sequences. This could impact algorithms that rely on strict ordering, like consensus algorithms or conflict resolution mechanisms.

Future Implications:

Incorporating indefinite causal order into computer science, particularly in quantum computing and communication systems, might challenge foundational assumptions about how programs execute, how data is shared, and how systems coordinate. Researchers are beginning to explore how these ideas could lead to new architectures that embrace uncertainty or non-linearity in causality, pushing beyond the limits of classical synchronization methods.

10.6.5 Compare to Lingua Franca

Edward Lee's *Lingua Franca* (LF) and the *Reactors* programming model are built around very different assumptions about time and causality than those implied by *indefinite causal order* in quantum mechanics. To explore the comparison, let's break down how each framework approaches time and causality and where they diverge.

1. Lingua Franca and Reactors: Assumptions About Time

- **Deterministic Logical Time:**
  Lingua Franca (LF) is designed to deal with time explicitly in distributed systems and cyber-physical systems (CPS). Its model assumes **deterministic logical time**, meaning the sequence of events is well-defined and unambiguous. The idea is to provide a clean, deterministic model where events are processed according to a well-ordered timeline.
  In the *Reactors* programming model, time is also central. Events are triggered in response to other events based on strict logical dependencies and causal relationships. Reactors operate under a causality principle, where one event triggers another, creating a deterministic flow of information. LF provides a way to manage this using **timestamps**, ensuring that events execute in a known order, even in distributed systems where physical time (real-world clock time) may vary.
  This focus on **logical determinism** means that in LF, all participants in a distributed system have a consistent understanding of the event ordering, even if they are physically separated. The system explicitly synchronizes on logical time to ensure causal relationships are respected.

2. Indefinite Causal Order:

- **Causal Ambiguity:**
  Indefinite causal order is fundamentally different. In quantum mechanics, particularly in processes like the *Quantum SWITCH*, events can occur in a superposition of orders. There is no clear distinction between "before" and "after" for certain events. This is because quantum mechanics allows for a superposition of states, which can include superpositions of different causal orders. As a result, the causal relationships between events can be indefinite or non-deterministic.
  In the framework of indefinite causal order, the assumption of **deterministic logical time** breaks down. The sequence of events might not be clearly defined until some measurement or interaction occurs, and the system could be in a state where causality itself is undefined or ambiguous. This introduces a fundamental uncertainty about which events caused others, contrary to the deterministic approach taken by LF.

3. Comparing Assumptions on Time and Causality:

- **Lingua Franca** assumes that:
  - Time is deterministic, meaning every event has a clear cause and effect, and that logical time is the primary tool to ensure consistency in distributed systems.
  - Logical causality can always be enforced and preserved through careful design of time-triggered reactions and event-driven computations.
  - The real world might have uncertainties in physical clocks (due to relativity or synchronization issues), but **logical time** can compensate for those discrepancies and create a shared temporal framework that all parts of the system agree upon.
- **Indefinite Causal Order**, in contrast:
  - Rejects the idea that all events can be ordered definitively. There may be no clear "cause" or "effect" in certain quantum processes.
  - Embraces the possibility that causality itself can be indeterminate, allowing for events to be in superpositions of different causal orders.
  - Cannot be modeled with strict logical timestamps because the very concept of time order may be undefined until measurement occurs.

4. Where Lingua Franca Works and Where it Struggles:

- **Lingua Franca's conveyor belt analogy:**
  In LF, time is much like a carefully managed conveyor belt. Events flow forward in time, and the system is designed to ensure that no matter how distributed or asynchronous the components are, they will agree on the causal order of events. This works beautifully in systems where deterministic behavior is crucial, like in CPS, real-time systems, or distributed computing where timing guarantees are needed.
- **Indefinite Causal Order breaks this model:**
  The assumption behind LF that all events can be deterministically ordered along a logical timeline fundamentally breaks down when applied to scenarios involving indefinite causal order. Quantum systems with indefinite causality cannot be modeled using LF's deterministic logical time, as there may not be any single "correct" timeline.

5. The Conveyor Belt vs. Forked Paths:

- **Lingua Franca:**
  The conveyor belt metaphor works well for LF. The belt may be fast or slow, but it always moves in a clear direction, and events are placed in a strict order on it. Logical time is like a regulating mechanism that ensures the belts of different agents are synchronized in the right sequence.
- **Indefinite Causal Order:**
  Instead of a simple conveyor belt, imagine a system where multiple belts can split, re-merge, and sometimes create loops. Events may occur in superpositions of these different belts. When you try to observe them, you might find that two events could have happened in either order, or perhaps simultaneously in different orders depending on how you measure.

6. Conclusion:

- **Lingua Franca and Reactors** are built on the classical assumptions of time: causality is strict, logical, and deterministic. Events can be ordered using logical time, and causality can always be respected across distributed systems. LF succeeds in providing a structured, well-defined temporal model that works in a wide range of practical applications.
- **Indefinite Causal Order** challenges the very foundation of these assumptions. In quantum systems with indefinite causality, events don't necessarily have a strict temporal order. Causality becomes probabilistic or even undefined until observed. This fundamentally

contradicts LF's reliance on determinism and logical timestamps. If LF were to be applied in a system where causal order is indefinite, the model would need to be rethought to accommodate the ambiguity and superposition of events.

The core difference between Edward Lee's Lingua Franca and quantum-based indefinite causal order lies in how they treat time: LF enforces strict, deterministic event ordering, while indefinite causal order introduces a probabilistic, non-linear flow of events where time and causality can be ambiguous and undefined.

10.6.6 Compare to Lamport Clocks

Lamport's notion of **logical time** and the concept of **indefinite causal order** represent two very different approaches to understanding time and causality in distributed systems and quantum mechanics, respectively. Let's compare these concepts in detail:

1. Lamport's Logical Time:

Lamport introduced **logical clocks** to address the problem of ordering events in distributed systems, where physical clocks cannot be perfectly synchronized due to the limits of speed, network latency, and other issues. Logical time doesn't rely on actual clock time but instead on the **happens-before relation**, which captures the causal relationships between events. The main features of Lamport's logical time are:

- **Happens-Before Relation ($\rightarrow$):**
  If event A causes event B (e.g., by sending a message), we say $A \rightarrow B$. This relation defines the causal structure in the system.
- **Event Ordering:**
  Every process maintains a logical clock. When an event occurs, it increments its clock and attaches this timestamp to any message sent. When another process receives the message, it updates its logical clock to reflect that the event has already happened, ensuring that causality is respected.
- **Total Ordering:**
  While logical time can establish a partial ordering of events based on causal relationships, Lamport's logical clocks do not give a complete global time ordering. However, vector clocks and other mechanisms can extend logical clocks to achieve more precise causal tracking.

The key idea is that Lamport's logical time helps enforce **causal consistency** across distributed systems, ensuring that events are ordered in a way that respects causal relationships between them. However, the system still assumes a definite sequence of events—either event A happens before B, or B happens before A.

2. Indefinite Causal Order:

Indefinite causal order, originating from quantum mechanics, breaks the classical assumption of definite event ordering. In certain quantum processes, events can exist in a **superposition of different causal orders**. This means:

- **No Definite Ordering:**
  Two events, A and B, can occur in a superposition of different sequences. It's not clear whether A happens before B or vice versa. Both possibilities can exist simultaneously until an observation is made.
- **Quantum SWITCH Example:**
  A quantum protocol like the *Quantum SWITCH* allows two operations to be performed in such a way that the order in which they occur is not definite. For example, A might influence B and vice versa, but the precise causal sequence is only determined probabilistically upon measurement.
- **Causal Superposition:**
  Indefinite causal order challenges the classical notion of time and causality by allowing for events that don't have a single, well-defined causal relationship. This differs radically from classical models like Lamport's logical time, where the goal is to create a definite order for every event based on causal relationships.

3. Comparison of Causality:

- **Lamport's Logical Time:**
  In distributed systems, **causality is explicit** and must be preserved. Event A either happens before or after event B, and Lamport's logical clocks enforce this by ensuring all processes agree on the causal order of events, even in the absence of synchronized physical clocks. The aim is to create a consistent and well-defined timeline.
- **Indefinite Causal Order:**
  In quantum mechanics, **causality can be indefinite**, and events can exist in a superposition of causal orders. This is fundamentally different from Lamport's approach, where causality is strict and must be maintained. In quantum systems, there may not be a clear, observable sequence of events until they are measured.

4. Time and Event Ordering:

- **Lamport's Logical Time:**
  - Relies on the happens-before relation to **preserve a well-defined, consistent causal order** between events.
  - **Logical timestamps** are assigned to events based on causal dependencies, so we can always say A happened before B, or B

happened before A.
  - The focus is on ensuring causal consistency, even in asynchronous, distributed systems.
- **Indefinite Causal Order:**
  - Time is not well-defined. Events can occur in a **superposition** of causal orders, meaning A and B might have happened in both orders simultaneously, and there is no definite causal relationship until a measurement is made.
  - The classical notion of time, where events occur in a linear, ordered sequence, doesn't apply. Instead, causality becomes **probabilistic** and only resolves upon observation.
  - This contrasts with logical time, which **enforces order** regardless of real-time discrepancies.

5. Applications in Distributed Systems vs. Quantum Systems:

- **Lamport's Logical Time:**
  Works well in distributed systems where ensuring a consistent, agreed-upon event ordering is critical. It helps manage concurrency, ensure that messages and actions are causally related, and maintain consistency in asynchronous environments. Lamport's clocks can provide a **deterministic causal structure** in environments where physical time cannot be relied upon.
- **Indefinite Causal Order:**
  Applies to quantum systems where the classical notion of causality and time does not hold. This idea is particularly useful in **quantum communication** and **quantum computing**, where indefinite order can be exploited for computational advantages (e.g., the *Quantum SWITCH*). It introduces a level of **uncertainty and flexibility** that cannot exist in systems governed by logical time.

6. Fundamental Differences in Handling Time:

- **Lamport's Logical Time:**
  Assumes a **linear, forward-moving flow** of events, where causal relationships are always definite and traceable. Time, in this context, is deterministic, even though it may be logical rather than physical.
- **Indefinite Causal Order:**
  Time can be **non-linear and non-deterministic**. Events may not have a clear before/after relationship until they are observed, meaning that causality is not well-defined in the classical sense. Time is less a forward-moving arrow and more a **probabilistic, superposed system**.

7. Summary:

- **Lamport's Logical Time** provides a **definite ordering of events** in distributed systems, ensuring that causality is preserved and that events follow a strict happens-before relationship. It operates within a classical, deterministic framework, where every event has a well-defined place in the timeline.
- **Indefinite Causal Order** introduces the possibility that causality is not always fixed. In certain quantum systems, events may not have a clear ordering, and time is **probabilistic** rather than deterministic. This concept challenges the very foundation on which logical time is built, as it allows for causal ambiguity and non-determinism.

In short, Lamport's logical time is a deterministic tool designed to enforce causality in distributed systems, whereas indefinite causal order belongs to the realm of quantum mechanics, where causality itself can be in superposition. The two are fundamentally at odds, as one imposes strict order while the other allows for causal uncertainty.

10.7 The Illusion of Simultaneity with Perfect Atomic Clocks

From ./AE-Specifications-ETH/standalone/Andromeda.tex

The relativity of simultaneity implies that two observers moving relative to one another can disagree on what events are happening "right now" at distant locations. Even if both observers carry perfectly synchronized atomic clocks, their determinations of simultaneous events at remote locations (e.g., the Andromeda Galaxy) can differ dramatically.

10.7.1 Time Shift Due to Relative Motion

Let v be the relative velocity between two observers, and D the distance to a distant object (e.g., a galaxy). The relativity of simultaneity predicts a difference in the perceived "now" at the remote location:

$$\Delta t \approx \frac{vD}{c^2}, \tag{10.1}$$

where c is the speed of light.

Examples

- For walking speed ($v = 1.39\,\mathrm{m/s}$) and $D = 2.5$ million light-years (Andromeda):

$$\Delta t \approx \frac{1.39 \times 2.365 \times 10^{22}}{(3 \times 10^8)^2} \approx 4.2 \text{ days}$$

- For hypersonic flight ($v = 1700\,\mathrm{m/s}$):

$$\Delta t \approx \frac{1700 \times 2.365 \times 10^{22}}{(3 \times 10^8)^2} \approx 5170 \text{ days} \approx 14 \text{ years}$$

10.7.2 Atomic Clock Precision

Modern optical lattice clocks can achieve stability better than 10^{-18}, corresponding to an error of less than 1 second over 30 billion years. Over a day (86400 s):

$$\delta t_{\text{clock}} = 10^{-18} \times 86400 \approx 8.64 \times 10^{-14} \text{ seconds} \tag{10.2}$$

THIS IS UTTERLY WRONG

It mistakes Gaussian variations in noise for time intervals. Mathematics doesn't work this way, The problem is (with $t \pm \delta t$) is not an interval. It's a belief in absolute simultaneity.

This is less than a femtosecond, utterly negligible compared to the relativity-induced differences in simultaneity.

10.7.3 Diagram: Disagreement Despite Synchronized Clocks

No degree of 'precision' or 'disciplining' of clocks will enable us to 'synchronize time".

Simultaneity is impossible in theory. It will therefore be problematic in practice. Physicists know this. Computer scientists have yet to discover relativity and quantum mechanics.

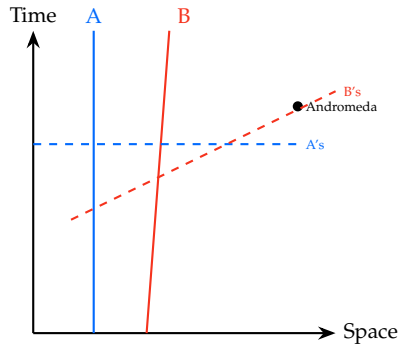


Figure 10.2: Two observers disagree on simultaneity with Andromeda events.

10.7.4 Conclusion

Perfectly synchronized atomic clocks do not resolve disagreements about simultaneity at distant locations. Such disagreements are a feature of spacetime itself in special relativity — not of timekeeping imprecision.

10.8 Zero: The Natural Number that Isn't

From commented out text in ./AE-Specifcations-ETH/standalone/Zero.tex

Zero is arguably the most fascinating number in mathematics, occupying a uniquely paradoxical position in our number systems. It serves as both a placeholder and a quantity in its own right, a concept that revolutionized mathematics when fully developed. Yet despite its fundamental importance, zero's classification remains a point of contention, particularly regarding whether it belongs among the natural numbers.

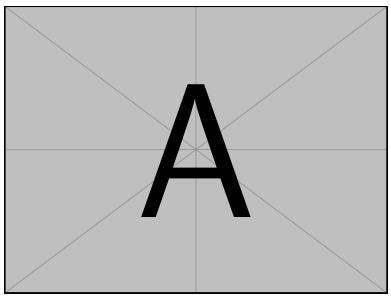


Figure 10.3: Various historical representations of zero across civilizations.

The question of zero's status as a natural number isn't merely semantic—it reflects deeper mathematical principles and has practical implications for how we define foundational concepts.

10.9 Historical Development

The concept of zero emerged independently in several ancient civilizations. The Babylonians used a placeholder symbol for zero around 300 BCE, while the Mayans developed a complete zero symbol around 350 CE. However, it was in India where zero was first treated as a proper number, with Brahmagupta establishing formal arithmetic rules for zero in the 7th century CE.

Brahmagupta's rules included defining $a - a = 0$ and $a \times 0 = 0$, but he struggled with division by zero, a problem that continues to challenge students today.

Mathematicians throughout history have disagreed about zero's classification. The ancient Greeks, whose work heavily influenced Western mathematics, had no concept of zero as a number. When European mathematics eventually incorporated zero, disagreements about its nature persisted.

10.10 Set-Theoretic Foundations

In modern mathematics, natural numbers are typically defined using set theory. The two predominant approaches yield different results regarding zero:

1. **Von Neumann ordinals**: Define natural numbers recursively where $0 = \emptyset$ (the empty set), $1 = \{0\} = \{\emptyset\}$, $2 = \{0, 1\} = \{\emptyset, \{\emptyset\}\}$, and so on. In this construction, zero is the first natural number.
2. **Zermelo ordinals**: Define 1 as $\{\emptyset\}$, 2 as $\{\{\emptyset\}\}$, etc. Here, counting begins at 1, excluding zero from the natural numbers.

The choice between these constructions reveals a fundamental question: Is mathematics primarily about counting (starting at 1) or about formal structures (where starting at 0 offers advantages)?

This fundamental distinction reflects different mathematical perspectives on what counting means.

10.11 Mathematical Arguments

Several compelling mathematical arguments support excluding zero from the natural numbers:

1. **Etymology and intuition**: "Natural numbers" traditionally refer to the counting numbers used in everyday life. When counting objects, we begin with one, not zero.
2. **Multiplicative properties**: The natural numbers form a monoid under multiplication with identity element 1. Including zero breaks this structure since zero lacks a multiplicative inverse.
3. **Division**: In the natural numbers excluding zero, division (when defined) always yields a unique result. Including zero introduces complications, as division by zero is undefined.

4. **Induction principle**: Mathematical induction typically starts with a base case of $n = 1$, implicitly excluding zero from consideration.

10.12 Notational Clarity

To avoid ambiguity, mathematicians have developed notational conventions:

- $\mathbb{N}$ or $\mathbb{N}_1$: Natural numbers starting from 1
- $\mathbb{N}_0$ or $\mathbb{N} \cup \{0\}$: Natural numbers including zero

10.13 Practical Implications

Whether zero is included among the natural numbers has substantial implications in various mathematical contexts:

- In combinatorics, excluding zero aligns with counting principles (you can't have zero of something when counting discrete objects)
- In number theory, including zero simplifies many formulations
- In computer science, zero-based indexing (starting array indices at 0) has proven advantageous for algorithm implementation

10.14 The Deeper Meaning

The debate about zero's status reveals a profound truth: mathematical classifications aren't discovered in nature but constructed by humans based on utility and consistency. The question "Is zero a natural number?" ultimately depends on the mathematical context and purpose.

This ambiguity isn't a flaw in mathematics but a reflection of its adaptability. Mathematical structures can be defined in different ways to serve different purposes, and these definitions are judged by their usefulness and elegance rather than absolute correctness.

Zero remains the bridge between positive and negative numbers, neither fully belonging to either domain yet essential to both. Perhaps this liminal position is precisely what makes zero so mathematically powerful—it stands at the boundary, connecting different mathematical realms.

The natural numbers may have begun as simple counting tools, but mathematics has evolved into a sophisticated framework where even our most basic numerical concepts reveal surprising complexity. Zero's contested status reminds us that mathematics, despite its reputation for certainty, contains fundamental questions whose answers depend on perspective.

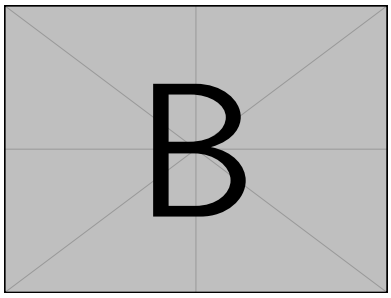


Figure 10.4: Visualization of the multiplicative monoid structure of natural numbers without zero.
The International Standards Organization (ISO 80000-2) defines $\mathbb{N}$ as starting from 1, while many computer scientists and set theorists prefer to include 0.

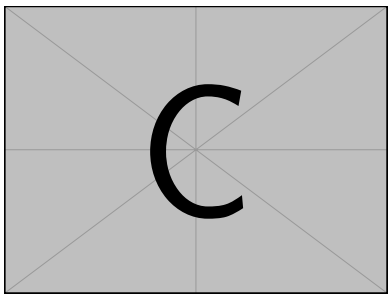


Figure 10.5: Example of zero-based indexing in computer programming.

11. Requirements and Use-Cases

11.1 Open Atomic Ethernet (OAE) Requirements

11.1.1 Core Technical Architecture

1. **Lossless / Near-Zero-Loss Data Plane.** Hardware flow-control or cell/TDM framing to eliminate random drops that invalidate atomic protocols.
2. **In-NIC Atomic Verbs.** One-RTT line-rate operations (e.g. CAS, multi-word commit) at $\geq$100 GbE serialization latency.
3. **Deterministic Scheduling.** Optional mesochronous or slot-based service for bounded latency (AI & HFT workloads).
4. **Reference-Free Causal Ordering.** Protocol must avoid dependence on globally synchronized clocks; use causal trees or hybrid logical clocks.
5. **Self-Stabilizing Control Plane.** Automatic convergence after arbitrary transient faults, no external reset.
6. **Composable Security & Isolation.** Native hooks for side-channel detection and optional crypto/QKD overlays without mandatory encryption.

11.1.2 Open Software Stack

1. **Open-Source Reference NIC RTL.** Minimal FPGA bit-file implementing OAE atomics; EFVI/Libfabric-compatible.
2. **Portable Verbs Library.** Unified C/C++/Rust API usable beneath Derecho, Raft, Aeron, Kafka, etc.
3. **Self-Healing Membership Service.** Gossip bootstrap $\rightarrow$ leader-based virtual synchrony (Derecho-style) with robust failure detection.
4. **Lightweight Management Telemetry.** Binary or gRPC schema (REST/Redfish only for out-of-band).

11.1.3 Research & Validation

1. Formal proofs for atomic commit without global time, gradient-clock bounds, and self-stabilizing recovery.
2. Tail-latency & failure-injection benchmark versus RoCE, DPDK, AF_XDP, Ultra-Ethernet, Tesla-TTP.
3. Hybrid quantum/classical shim for future Q-NICs.

11.1.4 Ecosystem & Governance

1. Public problem-statement registry (GitHub).

2. Sub-groups: Data plane; Control plane; Timing; Security; Business & Market.
3. External expert engagement (Rachee Singh, Shlomi Dolev, Roel de Vries, Chris Batten, Chris Rossbach, Tanya Zelevinsky, *etc.*)
4. Dedicated business-case work-stream quantifying TAM versus incumbent RDMA/RoCE.

11.1.5 Near-Term Deliverables (next 3–6 months)

#	Deliverable	Owner	Target Date
1	OAE Problem Statement v1.0 (15 pp)	Draft team	31 May 2025
2	FPGA NIC demo (64 B atomic commit @100 GbE)	Data-plane SG	30 Jun 2025
3	OpenAPI verbs library (C & Rust)	Software SG	15 Jul 2025
4	Self-stabilizing membership prototype	Control SG	15 Aug 2025
5	Market & cost model white-paper	Business SG	30 Aug 2025

Draft for discussion—please annotate, refine, and assign owners at the next OAE Working Group call.

11.2 Ethernet 2025 Requirements

11.2.1 Scope

- **Target deployments**:
  - Chiplet–to–chiplet interconnects *within a single package*.
  - In–rack, short–reach data-centre fabrics (one rack or a tightly-coupled pod).
- **Out of scope (initial release)**: traditional multi-rack Clos networks, WAN, metro, access, fieldbus and automotive links.

11.2.2 Functional Requirements

1. **Deterministic / Bounded-Latency Delivery**: provide end-to-end latency and jitter bounds tight enough for real-time AI training, HFT and chiplet coherency.
2. **Reliable Message Primitive of Bounded Size**:
   - Atomic "send → deliver" unit (frame or flit) *either arrives whole or is discarded*.
   - Optional acknowledged mode (*reliable-mode*) with automatic local retransmit; must be switch- and NIC-assisted but not mandatory for every packet.
3. **Timeout-And-Retry (TAR) Elimination in Fast Path**: protocol state machines must not rely on software timeouts for forward progress under nominal conditions.
4. **Autonomous Discovery & Configuration**: zero-touch bring-up, addressing and link training inside a rack/package; no DHCP or manual topology files.
5. **Packet-Processing Programmability**:
   - Data-plane micro-ops exposed at Layer 2½ (e.g. P4-like table, eBPF, or HDL modules) for in-line reduction, aggregation, atomics.
   - Configuration must be verifiable and sandboxed.
6. **Minimal Hardware Feature Set for Higher-Level Semantics**: supportive of virtual synchrony, one-sided RDMA, collective (AllReduce) and OLTP commit protocols, but *does not bake them in*.
7. **Classical Quantum Interworking Hooks**: reserve opcode(s) and timing model for future quantum-clock/entanglement assistance without constraining today's PHY/MAC.

11.2.3 Performance Requirements

1. **Latency Target**: single-hop store-and-forward <100 ns; cut-through path <50 ns (ASIC to ASIC).
2. **Throughput Target**: scale line-rate from 200 Gb/s (first silicon) to 1.6 Tb/s per lane within ten-year roadmap. Prefer parallel lanes / wavelengths over ever-higher SerDes clocks.
3. **Transactional Round-Trip**: single sender → single receiver ack <300 ns (in-rack worst-case).

11.2.4 Reliability, Resilience & "Atomicity"

1. **Loss / Corruption Handling**: per-hop FEC optional; end-to-end CRC > 2^{32} for silent error budget in AI workloads.
2. **Membership / Fail-fast Signalling**: hardware assist for fast liveness vote (virtual-synchrony–ready).
3. **Rollback Support**: hardware tag for speculative packets so software can discard/-commit without draining pipeline.
4. **Graceful Degradation**: mandatory congestion / buffer health telemetry; receivers own promise to detect and mitigate incomplete transfers.

11.2.5 Security

- **"Full-trust domain" assumption inside package/rack**, but hooks for Zero-Trust when links exit the trust boundary.
- Inline MACsec-lite profile ≤64 ns budget, key-roll without traffic pause.

11.2.6 Energy & Physical Layer

1. **Energy per bit**: <2 pJ/bit at 200 Gb/s; target sub-pJ with photonic lanes or advanced copper (e.g. TeraDSL / PAM8).
2. **Clocking**: support distributed, low-skew clock (photonic or electrical) to reduce SERDES power and jitter.
3. **Media**: copper <50 cm, twin-ax++ and/or ribbon fibre; hot-swap cages optional.

11.2.7 Manageability & Tooling

- Publish an open reference simulator (NS-3 or similar) and FPGA test-bed.
- Provide formal specs (TLA$^+$, DistAlg) and conformance suites for hardware vendors.
- Continuous telemetry stream: per-flow latency, congestion window, SERDES margin, temperature, link trust-state.

Note: The list reflects consensus *proposals* extracted from the chat; several items (e.g. mandatory rollback logic, quantum hooks) remain contentious and should be flagged for ballot.

11.3 Original Requirements

11.3.1 Charter:

11.3.2 Vision:

Ethernet has been the cornerstone of networking for over five decades, adapting to changing technologies, applications, and economic realities. New data center and edge opportunities—such as directly linking chiplet modules in ultra-low-cost meshes, achieving extreme low latency by executing application actions at hardware speed in the network interface, and applying these techniques to distributed updates of persistent storage or to distributed, application-level transactions-push beyond the boundaries Ethernet established in an earlier era. These needs cannot be met simply by extending Ethernet's historical capabilities.

The Ethernet 2025 Workshop and the Open Atomic Ethernet will look ahead to the next 50 years of Ethernet networking. It will focus on ways to achieve higher performance, guaranteed service, lower latency, fewer errors, and greater scalability. The idea is to make Ethernet omnipresent and capable of doing the job for everyone.

11.3.3 Scope:

Align what the network does with what the distributed application needs to accomplish. Build a world where the network is not allowed to create a mess the application must discover and then clean up, but rather must either ensure both sender and receiver (user space software endpoints in the distributed application) know a message was successfully sent and received, or only the sender "instantly" knows it failed, and no other node or part of the network knows that message ever existed.

This entanglement between sender and receiver, with no library, software, or hardware permitted to disturb that entanglement, is a fundamental change from the "throw it over the wall and hope it gets there" of the last 50 years. From a different perspective, a network message becomes a transaction which exhibits ACID properties: atomicity, isolation, and durability directly, and consistency because that message exists only between endpoints and is not visible to any other observer.

- Rethink Ethernet's core primitives: frame structure, error recovery, and flow control, in the context of Shannon Information Theory.
- ntroduce API's for programmable minimal compute capability at the NIC level to enable reversibility and support application-specific protocol behavior.
- Address the trade-offs between signals, noise and energy dissipation, building upon principles from Quantum Thermodynamics.
- New Fundamental multi-phase primitives: build state machines into the link itself to enable Network assisted transactions.
- Create a protocol framework that is as simple as Ethernet's original design but scalable to today's and tomorrow's use cases, including chiplet interconnects and AI accelerators.

11.4 Problem

11.4.1 Loss of Temporal Intimacy

1. Maintaining liveness and synchronizing processes in networks is a challenge when packets can be dropped, reordered, duplicated or delayed
   - Conventional networks require protocol stacks and applications to use timeouts and retries (TAR) to maintain liveness
   - This makes exactly-once semantics impossible and precipitates retry storms which lead to unbounded tail latency and transaction failure
     - This results in silent corruption of data structures and loss of data.
   - A problem seen in every distributed database for decades
   - These failures are not understood because customers (under NDA) may not publish results that embarrass vendors
   - The core issue is the "Unreliable" nature of Ethernet. It doesn't have to be this way
   - Nvidia has clearly seen this and understood how to create a "lossless" network with Infiniband
   - The Ethernet community sticks steadfastly to imposing multiple packets on the wire before expecting an acknowledgement.
   - The argument for this (50 years ago, in The Metcalfe + Boggs paper, and Bob Metcalfe's PhD thesis, is that bandwidth will be "throttled" too much if the sender has to wait for the acknowledgment before sending another packet.
   - These assumptions are no longer true in a world of rack-scale and chiplet infrastructure, even with 100Gbit 400Gbit, 800Gbit or 1.6Tbit lanes.
   - While the original Ethernet used "alternating slots" on the cable to transmit frames, Chiplet Ethernet must contend with minimum size (64-byte) frames on the wire

11.4.2 Network Faults Lead to Transaction Failure

- 80% of failures have a catastrophic impact, with data loss being the most common (27
- 90% of the failures are silent, the rest produce warnings that are unclear
- 21% of the failures lead to permanent damage to the system.
- This damage persists even after the network partition heals

11.4.3 Market

- (Current) Ethernet Has failed to resolve this issue with PFC or other inadequate congestion control algorithms.
- The world has chosen Nvidia.
- Ethernet has become unnecessarily fragmented.
- We need to bring the World back together under ONE Ethernet.

11.4.4 Topologies

Physical Topologies are (relatively) static in the Chiplet and Rack-Scale Infrastructure, but Logical and Virtual Topologies built on top of them are reconfigurable by Infrastructure software or operators, and programmable by application engineers within each secure enclave.

- Dedicated Circuit switching
- Link Level Negotiation
- May have cards from different manufacturers

11.4.5 Physical Link Protocol:

- All packets use the Alternating Bit Protocol (each packet is acknowledged before the next packet is sent)
- Failures may not be immediately apparent, but they will stop all activity on that link and will minimize the blast radius of failures.
- Receivers may NAK at the Information level, to indicate any kind or error, or to simply refuse the transaction asking the sender to forward it somewhere else.

11.4.6 Logical Link Protocol:

The roots of configuration trees don't have to be fixed to physical nodes. Graph Theory allows us to build logical and physical trees on top of physical trees (the groundplane).

11.4.7 Virtual Link Protocol:

The roots of application trees don't have to be fixed to logical or physical nodes. New results in graph theory allows applications to program their own tree structures on top of logical trees.

11.5 Frame Structure

Ultra-Low Latency is not possible without eliminating unnecessary fields and features in Ethernet's standard frame protocol

- Streams without Headers whenever Point to Point
- No Preamble
- No Inter-record Gap (IRG)
- No Destination mac address
- No Source mac address
- Both sides know who is sending who is receiving and what

11.5.1 Error Recovery

- Conventional Ethernet is a one-way Shannon Channel
- At Minimum, it needs "forward" Error detection (FED)
- Increased "one-way" resilience (OWR) omes from "forward" Error correction (FEC)

- FEC unnecessarily increases latency
- Atomic Ethernet is a bidirectional interaction Channel
- Alternating Causality protocol (ACP) (based on ABP) returns the first slice to the sender
- "This is what I heard you say"
- Eliminates the need for FED/FEC
- An entirely new way to think about Shannon Channels,

11.5.2 Flow Control

From Alan:

- Separate identity from address: A node can have a unique identifier (GUID) distinct from its address, which can be a small integer.

11.5.3 Secure Enclave

Security is impossible without renunciation of the source/destination mode of addressing.

Physical links are individually configured, identified, and provisioned, in seconds, reconfigured in milliseconds, and healed in microseconds.

MAC Addresses (48+ 48) and IP Addresses (32+32 or 128 +128) are susceptible anonymous attacks by byzantine players

- Adjacent (1-hop) links are semi-permanent. It takes physical action or decay processes to break them.
- Neighbor to Neighbor Networks (N2N) provide local privacy and eliminate global node identities

11.5.4 Leveraging Formal Methods

Formal Methods are essential to modern infrastructure verification.

11.6 Reconfigurable Topologies

11.6.1 Self-Adjusting Networks

- Infrastructure Reconfiguration
- Tenant Self-Reconfiguration

11.6.2 Application Enablement

11.6.3 Transaction Hyperparameter Space Features

- Help manage the rapidly changing state of art software environment.
- Help keep the hardware optimized for best performance, for example, by avoiding power overdraws
- Gerrymander workloads to balance power dissipation and reduce hotspots
- Provide a lighthouse for engineers, so they know where they are in the hyper perimeter space and how their model is doing versus how it could be doing theoretically.

11.6.4 Implementation Enablement

- Three key concepts around "Shannon information", the transfer of Shannon information between nodes, and how error recovery is done when that transfer fails or is interrupted, so that certain properties are protected (conserved quantities, "I know that you know that I know", Common Knowledge).
- We understand how to design and build everything above. Several implementations have been built and proven for the reversible transactions, in Javascript, in Rust, and in FPGA implementations of bits on the wire

- We use Wolfram Mathematica and the libraries and tools to explore the space of causal structures in ways that we expect to become standard industry toolchains for FPGA application fragments (specified by Petri nets, translated to rust, and compiled to verilog or VHDL.

11.7 FEEDBACK SO FAR

11.8 Anonymous1

11.8.1 Distributed Systems Problems

I asked Grok what are the unsolvable distributed systems problems. Here is its answer:

"In distributed systems, there are several fundamental problems that are proven to be unsolvable under certain conditions due to inherent limitations like latency, faults, and the lack of a global clock. These challenges arise from the nature of distributed computing, where multiple nodes must coordinate without perfect knowledge of each other's state. Here are some key examples:

1. Consensus in Asynchronous Systems with Faults (FLP Impossibility) The Fischer-Lynch-Paterson (FLP) impossibility result shows that in an asynchronous distributed system, it's impossible to achieve consensus (agreement on a single value) if even one process can fail by crashing. This is because, without synchronized clocks or bounded message delays, the system cannot distinguish between a slow process and a failed one. Any algorithm that tries to guarantee agreement will either risk never terminating or fail to handle all possible failure scenarios.
2. Byzantine Agreement with Too Many Faults In the Byzantine Generals Problem, nodes must agree on a decision despite some nodes behaving maliciously (e.g., sending conflicting messages). It's provably unsolvable if more than one-third of the nodes are faulty in a synchronous system. In an asynchronous system, it's even worse—Byzantine agreement becomes impossible with any faults unless additional assumptions (like trusted components or randomization) are introduced.
3. Perfect Failure Detection Detecting whether a node has failed or is just slow is unsolvable in asynchronous systems. Without a global clock or guaranteed message delivery times, a process waiting for a response can't determine if the other node crashed or if the network is delayed. This forces systems to rely on imperfect heuristics, like timeouts, which can lead to false positives or negatives.
4. Simultaneous Agreement on Global State (Snapshot Problem) Capturing a consistent global snapshot of a distributed system's state is tricky due to the lack of synchronized clocks and the possibility of in-flight messages. Chandy-Lamport's algorithm provides a solution under specific conditions, but without those (e.g., in a fully asynchronous system with unbounded delays), it's impossible to guarantee that all nodes agree on the exact same global state at the same logical "moment."
5. Two-Phase Commit with Network Partitions The Two-Phase Commit (2PC) protocol ensures atomicity across distributed nodes, but it's unsolvable in the face of network partitions. If the coordinator fails or becomes unreachable after sending "prepare" but before "commit," some nodes might be stuck indefinitely, unable to decide whether to proceed or abort. This is a practical manifestation of the CAP theorem, which states you can't have full consistency, availability, and partition tolerance simultaneously.
6. Exact Clock Synchronization Perfectly synchronizing clocks across distributed nodes is unsolvable due to variable network delays and relativity effects (though the latter is negligible in practice). Algorithms like NTP or Lamport's logical clocks provide approximations, but absolute precision is unattainable—there's always some drift or uncertainty.

   These problems are "unsolvable" in the sense that no algorithm can guarantee a solution in all cases under the given constraints. However, practical systems often work around them by relaxing requirements (e.g., assuming partial synchrony, tolerating inconsistency, or using probabilistic methods). For instance, Paxos and Raft sidestep FLP by assuming eventual message delivery, while Bitcoin uses proof-of-work to handle Byzantine faults probabilistically."

11.8.2 Funny, some of them are related:

- consensus is impossible, it was proved by Bernadette Charron-Bost forever ago
- group membership service is impossible because it is a consensus: all the processes must agree on who is in each group
- failure detection is impossible because it is a group membership problem: the processes must agree on who is dead and who is alive

  There were also detailed responses from Ken Birman which we can discuss.

11.9 Anonymous 1

I realize it's far too late to make any changes to this document before tomorrow morning. But I am not seeing a requirements document at the requirements level of abstraction, I am seeing a proposed/example definition of a novel link layer, without enough information to infer how the rest of what link layers traditionally do would be implemented. And my still-cloudy head doesn't yet grok what I am reading in the document.

My abstract thinking keeps coming back to the choices IBM/Ancor made in "Class 1" Fibre Channel (the selector channel, which never got market traction) 35ish years ago. Yes, I know, my employer was on the "Class 3" side of that bus war, and we won, and IBM lost, and Kumar made 3/4 of a billion dollars by selling his Brocade founder shares at the peak.

But as we look at the atomicity problem we're trying to solve (immediately knowing if a packet/message was received at its destination), not just for adjacent nodes but for any pairwise combination in a 2-D mesh of only somewhat bounded size, I keep coming back to the idea of start a communication level transaction (what Fibre Channel calls a SCSI Exchange) by reserving a path all the way through the mesh between the two parties, with the intermediate nodes functioning as pass transistors (actually, retimers) in the style of what Ethernet calls Layer 1 switches; doing the series of data movements, acknowledgements, etc between the two endpoints as if they were directly connected, and then tearing down the connection when the communication level transaction (the SCSI Exchange) is finished. Obviously we'd need setup and teardown at nanoseconds speed, not microseconds or worse.

Note that the real difference between "class 1" and "class 3" is that "class 1" is dedicated/connected, while "class 3" has all of the DReDDful behaviors of switches. "Class 3" still has to populate tables at least at both ends (for my employer's ASIC 30 years ago this was entries in the SCSI Exchange State Table) on a per communication level transaction basis; it's just the forwarding tables in the switches whose entries are longer lived.

If we were to do this, there would be two "bits on the wire" packet formats, one used in reserving the path, and a second used within the path once established. The protocol outlined in this document would be used in the second. My guess is the second would need a subchannel like handful of bits to identify which of (say) 16 paths from or through or to a node to send this on, which is remapped by a tiny data structure at each hop.

Separately

I am steadfast in my belief that each node maintaining a data structure describing the next-hop for each destination, and therefore gathering data only from its immediate neighbors, is an O(N) algorithm in each node for discovering and maintaining the forwarding tables (as implemented in eBGP, for example), while the "probe" concept is an $O(N^2)$ algorithm which will collapse as it scales past N=100, just as "spanning tree" did, and just as OSPF and IS-IS do. (Note that I am not an expert in this area, and that my characterization of $O(N)$ and $O(N^2)$are gross oversimplifications.)

11.10 Anonymous 2

11.11 All Mechanisms and Algorithms will be Self-Stabilizing

Shlomi Dolev presentation [02025-APR-09 Video Slides]

Self-Stabilizing Definition: [Wikipedia].

11.12 Anonymous 3

Maybe I am missing the context of the conversation, but I think you need to lead with the behavior you want to see, and from what perspective. it will anchor the piece into a root outcome.

Since it is really turtles all the way down, each individual reader perspective often only sees the turtle below them and above them. I found myself bouncing between application developer, compute designer, network operator, network hardware designer, and network standards perspectives...and I know my perspective is limited.

I'm fond of the idea, but without that grounding it feels a bit like you're trying to boil the ocean.

Here are my inline notes:

Thoughts on the introduction:

- Atomic acceptance at all nodes in a network will require a shift in the hardware model. Some form of separate item index, per interface, that would allow for signaling when a fan-in problem occurs and a buffer is overrun, the payload is lost but the header remains?

- OSI abstraction was to limit information required across the layers, sure it can be rewritten but the scale of information leakage is key, more on this later.

Efficiency of SAW:

- I'm confused by your clustered ACKs proposal. That's how windowing works.

- Your snakes proposal looks a lot like RSVP for the datacenter. This is being worked on within Broadcom for Jericho3/Ramon3. I'd love to see it outside of the hardware for network, but I'm not sure it could be fast enough. I'd also love to see it in a public standard rather than proprietary firmware. They aren't round-tripping the payload, but they are per-defining a path for a session and TDMing the links to guarantee the bandwidth.

- Assuming you are immediately reversing the data flow of the snake, prior to processing, your ACK will arrive at 'almost' the same time as a traditional ACK - if you opened your traditional window size to the size of the snake. Huge windows are pretty common, as are scaling windows tracking reliability. This is why you often see a sawtooth pattern in throughput. The difference would be waiting for the NIC to process and check the checksums on the traditional ACK. It could give you a more reliable ACK, but puts the onus of checking the payload on the original sender.

- Does the receiver proceed assuming good data or does it wait for a three-way handshake? No, you'd still need some form of FEC or CRC on the snake to be able to NACK if it doesn't match. Otherwise the failure scenario is rather painful.

- It might also push the operation ordering problem higher into the application, requiring it to know the order of atomic work asked for and ACK'd.

- Interface logic hardware scaling will be key, just as it is today. You'll need to scale what the interface logic can handle to the total possible dangling snakes, which leads you to a hard timeout on transmission based on hardware limits if nothing else.

Best Effort is not enough:

- "A full-duplex link is a logical grouping of two simplex links that are independent failure domains". This has been true for more than 20 years. In fact, they're no longer 2 simplex links but some arbitrary number of simplex links, often 4 each direction. Interface logic is making them appear as one to you. After the NIC packetizes it, the interface logic slices it up even further for sending. Again, all abstracted away and bounded. (Turtles all the way down.)

- You're talking about black and white failures, but those aren't the problem. Interface detection is pretty good. The grey failures are what kill network behavior....that and badly behaving apps. Parts degrade over time, someone stretches a fiber and cracks it just a bit, dust works its way into the interface, etc. Networks watch for light level changes and error rates, but not as well as they should. Every cycle not spent on sending traffic is latency.

- ACK/NACK, are you talking about it on a per-link basis or end-to-end? If it is per link then the input buffers just got a LOT bigger. Also, how do we account for the frame dropped from overrunning the buffer? Also, deep buffers mean latency. The first question in network hardware is 'drop or delay, which do you pick in contention?'

- "But in practice, these networks routinely violate packet causality through multipath routing (ECMP), queuing disciplines (EQDS), and load-balancing heuristics that scatter IP fragments across the network." You're going to need to back this up. Session-based load balancing has been the norm for 20 years, specifically to solve the ordering problem. To speed things up folks are starting to look at per-pack load balancing again, but it hasn't been common since the 90s.

—

Overall, it feels like a solid half of a proposal. I'm coming at it from a, "What network hardware do I design to be able to do this?" I know that's further down the line than you are yet, but there are some very fundamental gaps in the 'what', before i start thinking about the 'how'. I think anchoring this, as I mentioned at the top, will help focus that conversation and aid sharpening the 'needs' portion of this paper.

Pushing back the other direction, let me rehash an idea Mae posed:

90+% of this could be solved with a separate E2E timing estimation as a hard timeout. Either declared based on architecture or as a side-channel application like MSFT's pingmesh, define a hard timeout that can be leveraged by the applications. "I have not received ACK within TIMEOUT, therefore resend." If all 'snakes' have a UUID, duplication isn't an issue. A rolling hash where entries expire after an hour should easily cover that very managably.

Now let's talk information leakage (necessary information sharing?). Hearkening back to the line, "All models are wrong, but some models are useful." The problem with pointing to the black box of infrastructure and citing reliability failures at the higher levels is that it is failing to discuss the scaling problems within the black box. Information flow up from the black box to the application is reasonable because it is a bounded set of data. Application knowledge flowing into the black box is an unbounded set. Network latency, cost, and behavior is directly tied to the bounding of that set., otherwise no amount of hardware will accommodate the fan-in.

End of the day, speaking across perspectives will be key to driving this. Most everyone reading will only react to it from their own perspective, and they can be quite different. When I need to demonstrate the difference, I explain it like this:"Developers are artists, they are sculpting in code and in math. Infrastructure, on the other hand, is building a bridge out of bent and cracked tinker toys. Nothing is to spec, the lengths aren't exact, the holes aren't drilled at exact angles, but they still need to be able to move the Tonka trucks over them." That usually starts a larger conversation, but it is a good statement to anchor from.

11.13 [Anonymous 4]

11.13.1 QUESTION: What's different about Atomic Ethernet?

11.13.2 ANSWER:

There are some significant conceptual differences between the Atomic Ethernet `Link`[1] Protocol (AE) and traditional networking protocols in today's distributed microservices, key-value stores, and databases. For example:

- The protocol is *not* an asymmetric 'source-destination' protocol. But a bipartite 'token coherence protocol' – the ability of one side, to 'know' whether or not the other side has consumed (i.e., observed) the token. It is *not* a source-destination protocol as in traditional Ethernet. Local Addressing is by port and compass point scout packets. Global Addressing is rootward or leafward on trees.
- Messages are not 'resent'. If a failure occurs for any reason during the 'transaction' then the transaction is first reversed out ('unsent') and then re-applied[2].
- This is essential; any detritus[3] from the previously attempted transactions must be completely eliminated, such that, the next time the transaction is attempted, is 'the first time'.

This is in-contrast to the conventional 'forward-in-time only' protocols, whose effect is transformation from a previous state to an irreversible future state. Genuine (logical or physical) reversibility provides simpler and more guaranteed capabilities for error recovery that are not available to forward-in-time only protocols.

The new perspective, is that instead of the source (initiator) executing a state change in a forward direction in time, both parties are responsible for the successive forward, *and* the successive reversal of the atomic operation when an error occurs; not just one of them, with perhaps an incomplete knowledge of what state changes may have occurred in the target. Logical reversibility can be viewed as *time reversal*, only if (1) it is successive, and (2) all trace of the transaction has been removed with no evidence whatsoever that it occurred[4].

A central issue in the E2E debate appears to be whether or not the network is *lossless*; and the performance penalties for recovering from those packet losses. We summarize our position on packet loss thus[5]:

> In an event-driven system we convert a packet loss to a `Link` failure. The only failure we have to deal with is `Link` failure; which is recovered by another `Link`.

I.e. We treat packet loss (or corruption), not as an exponential complexity of failure recovery mechanisms, but as an opportunity to simply 'reverse time' so that the error never occurred. This new perspective is central to our ability to achieve exactly-once semantics, fast failover, and security guarantees.

[1] We distinguish between a conventional 'link', and the AE `Link`, with capitalization and a different type face.

[2] If we allow message retry, there are classes of faults that only require one round trip (we don't allow message retry). If we allow packet loss, we would need a message retry in the 2-phase protocol. If there are no faults at all, one round trip suffices.

[3] Even with sequential programs, once we have side-effects, we have many more opportunities to confuse ourselves, and this can get much worse with concurrency.

[4] This requires that the internals of the transaction must also be hidden from any kind of observation. until both sides have reached some system level threshold of knowledge needed for the promises expected.

[5] The Bill Jackson formulation.

11.14 Market for a Sea of XPU's are the *closed* hyperscalars

The FPGA NICs aren't cheap compared to a standard NIC. The only ones our use cases would work with were RISC-based Linux ones with, if not currently, at least planned, per-core (single-thread) performance close to single-thread performance of a Xeon core. Broadcom's Stingray II (shelved), Marvell's high-end Arm processor (never produced in volume), and Intel's Mt. Evans roadmap (never pursued for anybody other than hyperscalers). NVIDIA's BlueField 4 (not yet

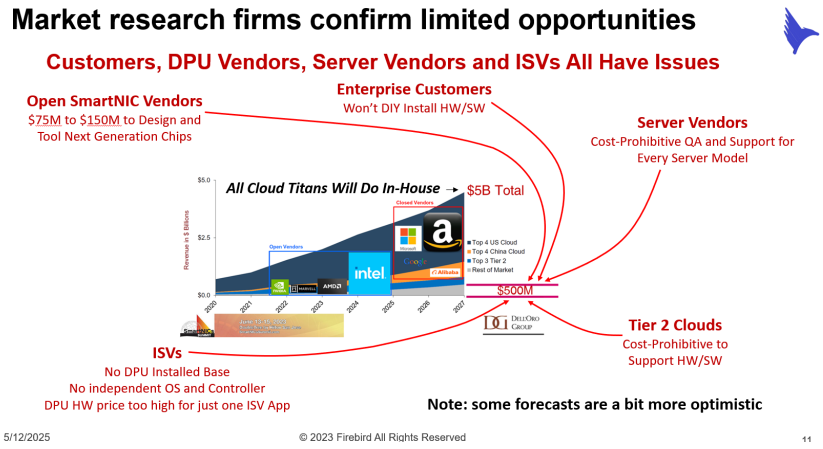


Figure 11.1: The SmartNIC Revolution died, long live the XPU Revolution. Courtesy: Harry V. Quackenboss

released – maybe shelved) looked good on paper.

The really high-volume prices of the SmartNICs in the BlueField-3 class were probably as low as $700 per unit, which I suspect is still less way than FPGA NICs.

For our use cases, price/performance was better on these types of SmartNICs because the data in flight nature of the workloads meant less DRAM/core, and the RISC versus X86 advantage of 70% in MIPS/Watt meant it made sense to offload those. However, if the SmartNICs couldn't keep up with he geometry shrink from 3nm -> 2nm and so on that the CPUs were on, the per-core performance and MIPS/Watt advantage evaporated.

———— Harry Quackenboss

Are these honking big, expensive smart NICs you're talking about? I think we can implement our protocol in a pretty simple FPGA. (It can be really simple if we do failure recovery with the CPU.) In fact, there was an FPGA implementation. John Lockwood may know the details.

Years ago we spoke to a NIC vendor. We could implement 99% of the protocol with small changes to their standard product. There was one corner case that required a change to their pico-code that was the showstopper for them. None of those changes would have been visible to customers not using our protocol.

———— Alan Karp

I'd say the future is highly interconnected processors that have specialized computing support, SmartNICs fall in the specialized com-

puting category if they have engines for things like encryption and compression, or something general purpose like an FPGA.

———— Kev.

Thanks for this perspective. I had thought that smart NICs everywhere was the wave of the future.

That being said, I'm not sure we need smart NICs for the basic protocol. I believe that the happy path can be handled by simple state machine logic in the NIC. Recovery from a broken link or node is probably too complex for that, so it would have to be handled by the CPU. Recovery would certainly take longer, perhaps ms instead of microseconds.

———— Alan Karp

It strikes me that among Kevin, Alan, and Hesham, there is no common understanding of what the minimum NIC feature set is that this future protocol suite would run on (or, for that matter, possible middle boxes such as L2 switches)

Adrian is current on this stuff, and I am not, but my understanding is that SmartNICs, to summarize, are used by AWS, Google, Microsoft, and Meta (and as a first approximation, Meta is less than 1/5 the size of the others), but each vendor controls the SW stack including host drivers, and firmware on the SmartNIC, and even at least some HW features. And, at least for now, none of the public cloud subset is making programmability available to customers.

I don't see how this effort is going to impact that, but maybe Adrian has a different opinion.

Other than niches (and compared to markets big enough to support new silicon at the rate CPUs evolve, even FPGA NICs are in my taxonomy, niches). NVIDIA's adaption of their DPU family to allow tuning for RDMA in AI and HPC clusters (that NVIDIA calls "SuperNIC") isn't really an exception in my view. There isn't a significant installed base. If you scrutinize the forecasts and separate out vendors obfuscating to situation by declaring their high-end server NICs as SmartNICs, none are forecast big growth in the future (again, excluding hyperscalers).

A conclusion I draw from that is that trying to create new broad industry standards is inconsistent with depending on hardware roadmaps for NIC features that aren't forecast to be big volumes (excluding the hyperscalers that are going to be difficult to influence.

Translation: if this effort depends on custom programmable NICs (SmartNICs) isn't going to lead to broad interest.

The sooner this group can come to a common understanding about whether SmartNICs are in or out of scope, the sooner the debate can be narrowed.

I personally would love to see SmartNICs be widely adopted. As Kevin intimates, there are a bunch of things one can do.

I just think unless and until factors beyond this group's ability to influence change, it's like trying to start a campfire with damp kindling in the rain. Smoke, with effort. But not much fire.

———— Harry

11.15 Our Unfair Advantage in Networking

Conventional Clos Networks assume a "flat" communication model, and an Any to Any (A2A) traffic pattern. What if we could take advantage of spatial (neighborhood) and temporal (synchronization) traffic patterns and provide ultra low latency in directly connected (switchless) networks?

11.15.1 ML/AI Inference on The Edge

While training is for the big boys, and often needs gigantic capital and operating budgets, inference can be carried out with more modest budgets, on far smaller networks, fewer GPUs and CPUs and is highly constrained by latency requirements of customers whose attention span has been trained by expectations of search engine response times.

These latency characteristics are also naturally serviced from the edge where the customers are, rather than the big centralized datacenters owned by the superscalars.

This provides many more market insertion opportunities for DÆDÆLUS' unique insights and unfair advantage as a software infrastructure company bringing new capabilities and value to the edge.

11.15.2 Local not Global Traffic Patterns

Datacenter Traffic Matrices are sparse and skewed, exhibit locality and are bursty. [Griner SIGMETRICS 2020]. They are far from arbitrary or random, which is often used as an justification for a "Flat" networks model in datacenters.

Reconfigurable networks [Dynamic Trees] allow to establish direct links between two frequently communicating pairs of racks in a datacenter, e.g., using digital micromirror devices.

11.15.3 Snapshots need Reliable Subtransactions

Both Transactions (MVCC) and AI/ML rely on snapshots being consistent. But they can't be made consisted using timestamps alone.

11.15.4 Spanning Trees under application control

11.15.5 Failure, isolation, healing: using local information only

11.15.6 Topology Awareness for Applications like AI/ML

11.15.7 Sequencing and clocks without timestamps

Timestamps are an unsafe ordering construct.

11.15.8 Our unfair advantage

- Eliminate timeouts & retries, which cause cascade failures
- Switches and routers not needed
- SmartNICs or Chiplet microdatacenters
- Easy to use (compared to conventional networks)
- Links carry only functional (not transit) packets
- More potential Bandwidth at the same cost
- Ultra Low Latency is lowest possible: governed only by the length of the connections
- Connections - Ours are are shorter. (2cm vs 2m vs 20m)
- Encryption not needed between adjacent nodes (Saves computation, time, and energy)
- Developers control their own application topologies
- No bandwidth amplification, either by Gossip protocols or timeouts and retries

11.15.9 Opportunities we Unleash

HammingMesh is Optimized specifically for deep learning workloads and their communication patterns.

> "Deep learning training uses three dimensional communication patterns and rarely needs global bandwidth. It supports extreme local bandwidth while controlling the cost of global bandwidth."

HammingMesh uses a 4x4 Tile, connected in a valency-4 (square) mesh, and overlays a Hamming code to provide a GEV coordinate system for routing. As with many coordinate systems, including the cartesian coordinate systems used in most Torus networks, they become extremely complex when they fail and we have to route around

failed links and nodes. This is not the case for DÆDÆLUS, where every node has an coordinate system with itself at the root. Because

The Strawman objection, by conventional network architects, uses "diameter" as a figure of merit (or demerit). Includes three misconceptions:

- Nodes are "fixed" in some Cartesian layout, nailed to the floor of the datacenter in some specific rack (our addressable entities are mobile)
- Hops are somehow related to latency in a fixed cartesian coordinate system, so the diameter gets worse as you scale (DÆDÆLUS exploits neighborhood locality)
- Execution entities are randomly placed throughout the datacenter — for a "flat network concept".
- Most AI/ML Traffic patterns are highly predictable. Once the pattern is established in the first few passes, the communication patterns can be exploited and the virtual trees can be optimized to match those patterns.

Bibliography

Norman Abramson. THE ALOHA SYSTEM: another alternative for computer communications. In *Proceedings of the November 17-19, 1970, fall joint computer conference on - AFIPS '70 (Fall)*, page 281, Houston, Texas, 1970. ACM Press. doi: 10.1145/1478462.1478502. URL http://portal.acm.org/citation.cfm?doid=1478462.1478502.

Aaron Bernstein, Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Deterministic dynamic maximal matching in sublinear update time. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 132–143, 2025.

Eric A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, page 7, New York, NY, USA, July 2000. Association for Computing Machinery. ISBN 978-1-58113-183-3. doi: 10.1145/343477.343502. URL https://doi.org/10.1145/343477.343502.

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Dale Woodford, Yasushi Saito, Christopher Taylor, Michal Szymaniak, and Ruth Wang. Spanner: Google's Globally-Distributed database. In *OSDI*, 2012.

Donald W Davies, Keith A Bartlett, Roger A Scantlebury, and Peter T Wilkinson. A digital communication network for computers giving rapid response at remote terminals. In *Proceedings of the first ACM symposium on Operating System Principles*, pages 2–1, 1967.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Siva-

subramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

Jim Gettys and Kathleen Nichols. Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65, January 2012. ISSN 0001-0782. doi: 10.1145/2063176.2063196. URL https://dl.acm.org/doi/10.1145/2063176.2063196.

L. Kleinrock and F. Tobagi. Packet Switching in Radio Channels: Part I - Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics. *IEEE Transactions on Communications*, 23(12):1400–1416, December 1975. ISSN 1558-0857. doi: 10.1109/TCOM.1975.1092768. URL https://ieeexplore.ieee.org/abstract/document/1092768.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978. ISSN 0001-0782, 1557-7317. doi: 10.1145/359545.359563. URL https://dl.acm.org/doi/10.1145/359545.359563.

Edward A. Lee, Soroush Bateni, Shaokai Lin, Marten Lohstroh, and Christian Menard. Quantifying and generalizing the cap theorem, September 2021. URL http://arxiv.org/abs/2109.07771. arXiv:2109.07771 [cs].

Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, July 1997. ISSN 0146-4833. doi: 10.1145/263932.264023. URL https://dl.acm.org/doi/10.1145/263932.264023.