1. History

Sections taken from ./AE-Specifications-ETH/sections/History.tex

1.0.1 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.

This is Saltzer 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.

1.0.2 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.

1.1 ALOHA

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

This section compares the classical ALOHA protocol with its improved variant, Slotted ALOHA, highlighting differences in their design, collision behavior, and efficiency.

1.1.1 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

??

1.1.2 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 1.1: Key differences between ALOHA and Slotted ALOHA.

1.1.3 Visual Comparison

Figure 1.1 shows a visual comparison of packet transmissions and collisions in both protocols.

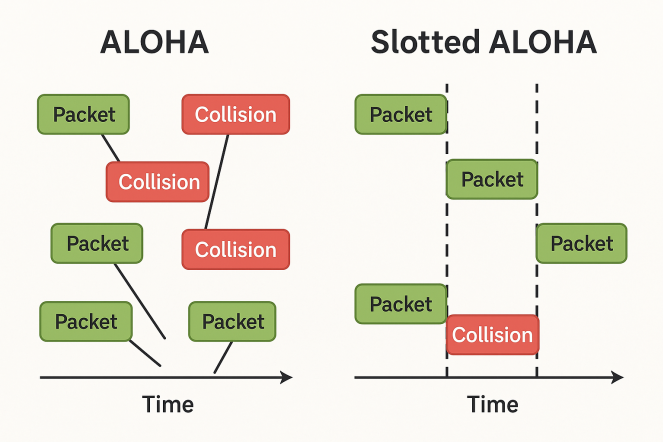


Figure 1.1: Collision behavior in ALOHA (left) vs. Slotted ALOHA (right).

1.1.4 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.

1.2 ATM

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

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.

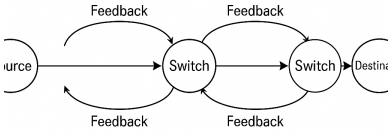
1.2.1 The Challenge: ATM's Dual Mandate

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.

1.2.2 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
opagate slowly end-to-end.

Figure 1.2: 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.

1.2.3 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.

1.2.4 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.

1.2.5 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.

1.2.6 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.

1.3 Ethernet

Sections taken from ./AE-Specifications-ETH/sections/BSW.tex

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.

1.3.1 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.

1.3.2 Reliable vs. Infallible, Unreliable vs, Fallible

Table **??** highlights the nuance. Priority Flow Control (PFC) can render an Ethernet link *loss-less* in steady state, but deadlock, mis-configuration, 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."

1.3.3 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.

1.3.4 Summary of Robert Garner's Equations and Assumptions

From ./AE-Specifications-ETH/standalone/Atomicity-ChatGPT-1.tex

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.

1.3.5 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).

1.3.6 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.

1.3.7 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.

1.3.8 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.

1.3.9 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).

1.3.10 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.

1.3.11 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.

1.3.12 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.

1.3.13 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.

1.3.14 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.

1.3.15 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.