Reinventing Reliability at Layer 2

Sahas Munamala, Paul Borrill, et. al.

April 14, 2025

***Abstract*—Reliability in distributed systems has long been delegated to applications already burdened with handling every imaginable failure. The OSI model encourages vendor independence and fosters rapid innovation at lower layers, enabling the modern internet—but at a cost. Reliance on the End-to-End (E2E) principle, which tolerates dropped, delayed, reordered, and duplicated packets while expecting applications to ensure correctness, worked when those applications ran on single machines. Today, datacenters are filled with retries and congestion control mechanisms, forming a fragile ecosystem where even minor changes can dramatically affect throughput and tail latency. We propose a Layer 2 acknowledgment scheme that treats packets as self-contained, bidirectional "snakes" carrying both data and acks. By rethinking reliability beneath the transport layer, this approach removes retransmission timeouts, and enables more predictable, fault-tolerant behavior—offering a compelling alternative to retry-heavy systems constrained by the E2E model.**

To ACK or Not to ACK? [1]

A focused summary of Robert Garner's key equations, assumptions, and reasoning—drawn from email threads—on ACK/NAK-based (reliable) protocols layered atop an *unreliable* Ethernet.

> "My perspective (as was Metcalfe's) is that it's not that Ethernet trades off higher performance for lower reliability, but instead pushes reliability up to the next higher level of protocols. Metcalfe realized that different applications over Ethernet will need/want different degrees of reliability. For example, it may not be a disaster if an occasional network status inquiry or voice packet is lost (former retransmit, latter interpolate), but it would be a disaster if, say, a monetary or database transaction packet were lost." —Robert Garner

Our argument: Our argument: A reliable Ethernet layer forms a solid foundation for building atomic operations directly into the network. When reliability is guaranteed at Layer 2, higher layers no longer need to rely on sender-side timeouts to ensure delivery—eliminating a major source of unbounded tail latency.

Application-send is an event in spacetime. If the information from that event is lost, corrupted, or silently dropped at Layer 2, higher layers cannot reconstruct it without incurring exponentially growing complexity. The illusion that upper layers can fully recover from such losses is a dangerous assumption—especially in databases and distributed systems, where failure to preserve atomicity or isolation leads to irreparable consequences.

We concur with Bill Lynch:

> "It is quite possible to use a higher-level protocol *on top* of an ACK/NAK protocol to achieve throughput approaching the connection's inherent capacity." —Bill Lynch

[1] With apologies to Hamlet.

Problem

For over five decades, engineers have managed the complexity of networked systems through layered abstractions—most famously the OSI model. This design has enabled vendor independence and the steady evolution of networking technologies. At the heart of this lies the end-to-end (E2E) principle, which delegates reliability to the application layer. This approach is built on the foundational assumption—rooted in Bob Metcalfe's 1970s calculations [1]—that the network itself should be best-effort and unreliable.

Ethernet's (and E2E's) success is carved in the history of computing – it enabled the first wave of network and internetwork technologies we call the internet. In this era, applications and services were mostly centralized with one machine owning coordination and durable state of the system. Even ServerNet networks by Tandem's Nonstop, the first money transfer systems, worked by passing the ownership of a transaction from one centralized server to another, removing any global coordination requirement. Around the year 2000, scale increased dramatically. Systems like eBay, Webvan, and AltaVista had to evolve from single-server architecture to massively distributed systems.

The E2E principle assumes smart endpoints and a dumb network, which worked when endpoints could coordinate state easily in one core system. But in distributed systems multiple endpoints need to coordinate state fast and accurately. This kind of coordination wasn't part of the original networking workload; Ethernet wasn't designed to guarantee that multiple nodes can stay in tight sync under high load and failure conditions. Meanwhile, database systems routinely struggle with communication failures that undermine their ability to

uphold the "ACID" properties—especially atomicity and isolation—once taken for granted [2], [3].

From a physics perspective, this problem resembles the contrast between correlations and signaling, the implications of the no-cloning theorem, the Two-State Vector Formalism, and Indefinite Causal Order [4], [5]. These concepts are foreign to most computer scientists and network engineers. Yet, with Bill Lynch's insights and contributions, we've uncovered classical analogs—mathematical constraints in reliable protocols—that mirror these quantum behaviors [6], [7].

Clinging to this approximation blinds us to the true nature of reliability failures at higher layers, rendering recovery—especially in distributed transactions—impossible [8]. Failures in atomicity and isolation are not just bugs to be patched; they are consequences of a deeper structural misunderstanding. (Consistency and Durability, we leave for another forum.)

> **Ethernet's primary asset is the large and thriving ecosystem that allows a multitude of vendors to contribute to a customer system. That ecosystem was grown by a free, open, and detailed system of standards provided first by the involved vendors and later by the professional standards association.**
> [Bill Lynch]

The Efficiency of Stop and Wait

Ethernet was originally designed to detect and retransmit collisions, with early models illustrating the transmission efficiency of the shared medium [1]. Stop-and-Wait (SAW) protocols can work on a shared medium like the original tapped coaxial Ethernet, but they become inefficient under bursty traffic conditions due to contention. The key characteristic of SAW is the next transmission is not started until the ACK for the previous transmission is received. This forms a dependency between transmission events in the sender and receiver's local-time, and enforces the strict ordering of events required for reliable transmission in one direction.

Inspired by Metcalfe's original efficiency model, we analyze a Stop-and-Wait (SAW) protocol over a modern point-to-point link. In this setup, a data packet of length P is transmitted in one direction, followed by an acknowledgment packet of length A sent back in the reverse direction over the same link. In a modern notation, the *effective capacity* E of an ideal channel (no packet loss) with those stop-and-wait characteristics is expressed as:

$$E \;=\; \left(\frac{S}{P}\right) \times \underbrace{\frac{T_p}{T_p + T_l + 2T_r + T_a}}_{\text{Bandwidth Degradation } M} \times C \qquad (1)$$

where:

- E = Effective capacity (perfect reliability, zero packet loss, no congestion, no queues)
- C = channel capacity (bits/s)
- S = payload (data) bits per packet
- P = total packet length (bits)
- A = total ACK length (bits)
- T_l = two-way link latency
- T_r = processing time between receiving events to triggering new events (reaction time)
- $T_p = \frac{P}{C}$, packet transmission time
- $T_a = \frac{A}{C}$, acknowledgement transmission time

A strict stop-and-wait (SAW) model, like the alternating bit protocol[9], dramatically throttles throughput. With a transmission rate of 10 Gb/s, transmitting a 64-byte packet takes about 51.2 ns and covers roughly 10.2 m of copper cable. However, on a 1 m copper cable, $T_l \approx 10\,ns$, and assuming a reaction time $T_r \approx 6\,ns$, the best degradation factor possible is $\lim_{T_a \to 0} M \approx 0.7$. At 100 Gb/s, transmission of 64 bytes takes 5.12 ns and 1.02 m of copper cable, worsening the degredation factor to $\lim_{T_a \to 0} M \approx 0.2$.

Supporting multiple packets in flight, increasing P, or reducing A can increase throughput; All options optimize the M factor. If, for example, one ACK can cover 10 packets, the ratio $\frac{T_a}{T_p}$ shrinks, and the effective one-way channel utilization can be optimized to approach 80–90% or more. SAW deniers claim the model is a fundamental performance bottleneck in networks, where the limited ability to keep the link fully utilized leads to significant inefficiencies in throughput, especially bandwidth increases. In the next section, we show a small perspective shift is all that is required to construct a SAW protocol without throughput decay.

A Change in Perspective

An ideal acknowledgment (ACK) scheme should maintain throughput while ensuring the closure of information within the bi-directional channel. Without predesignating one side as the server and the other as the client, operations must be identical on both ends [10].

To address this, we introduce a new way of thinking about packets in transit: not as discrete monolithic packets, but as snakes—logical self-contained entities that can span both directions of the link. A snake begins with a head that carries data, and as it moves through the channel, it represents a bi-directional exchange: the forward transmission and its eventual acknowledgment are unified within a single structure. But unlike a traditional packet, a snake is its own acknowledgment. As it completes a round trip through the link, each side performs a reversible transformation—such as a bitwise inversion or slice reversal—that, after two passes,

returns the data to its original form. This restoration marks the successful completion of the exchange: the acknowledgment is not a separate packet but an emergent consequence of the snake's return.

The snake may not occupy the full length of the link; in fact, its length relative to the round-trip time is a central factor in overall throughput. But wherever the snake exists in the link, it serves as a live representation of in-flight data and its in-progress acknowledgment, tied together in both space and time.

In the setup illustrated in Figure 1, a simple INVERT operation is applied at each endpoint. Because this operation is its own inverse, it restores the original data after two passes through the loop. The receiver does not buffer or interpret the payload; instead, it immediately applies the reversible transformation and returns the result along the unoccupied return path. The acknowledgment is thus not a distinct payload, but is fully contained within the act of circulation itself—the presence and position of the snake within the link.

Additional snakes can be introduced to fill unused portions of the link and improve total throughput. Crucially, these new snakes do not interfere with existing ones. The bandwidth degradation of using n snakes *without any buffering* is governed entirely by the ratio of bit-snake-time to the total round-trip-bit-time of the link:

$$M = \frac{nT_p}{T_l + 2T_r},\ n \in \mathbb{N}$$

Only an integer number of snakes can fit within the round-trip cavity of the channel. The numerator scales linearly with the number of snakes and their individual payloads T_p, allowing M to grow beyond that of a single stop-and-wait packet

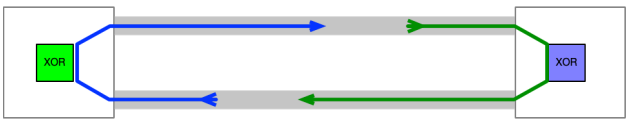


Fig. 1. On wire longer than a single bit-snake, multiple snakes can occupy the link to maximize link utilization.

This approach can be viewed as a physical reimagining of "multiple packets in flight." While each snake carries new data in its head, the total number of unacknowledged bits in flight is still fundamentally constrained by the round-trip latency of the link. To preserve the atomicity of the reversible transformation, both ends of the link must agree on a fixed frame size—effectively presenting the transmission and acknowledgment as a single indivisible event to higher layers.

However, this model introduces a natural limitation: if the link is not long enough to accommodate another full snake, the unused segment remains idle, wasting potential throughput. And as the link length approaches zero, the snake has nowhere to go. It gets crushed against the walls of the link, and the unbuffered bit-snake dies. Its only chance at survival is to store parts of itself—its vertebrae—in logic on either end of the link.

To make this possible, we reimagine the snake not as a single, monolithic packet, but as a chain of eight 8-byte vertebrae—aligned with the slice width of typical 10 Gbs and 100 Gbs SerDes designs. Each side now maintains a fixed 8×8-byte FIFO, just large enough to host a full snake in the worst case, as depicted in 2. At minimum cable length, the system behaves like a register transfer: both sides load a complete snake, apply the reversible transformation, and return it to the sender to complete the acknowledgment cycle. Bitwise inversion can be replaced with a slice-level operation such as bit reversal, still preserving the two-pass semantics.

As the cable length increases, more vertebrae fit in flight, and the FIFO rarely fills. Each side only holds the minimum number of slices needed to maintain the illusion of a continuous snake. We dynamically "stretch the wire" until the bit-snake-time equals the round-trip-bit-time of the link.

By doing so, we rescue the snake from its early death. It now lives in both the channel and the interface logic, seamlessly weaving between the two. The result is a fully saturated link with zero waste, built-in acknowledgments, and no need for complex buffering. The bandwidth degradation factor reaches its ideal:

$$M = 1$$

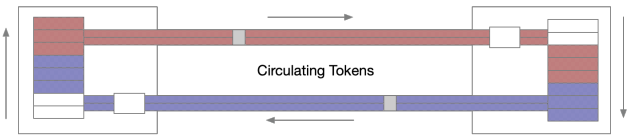


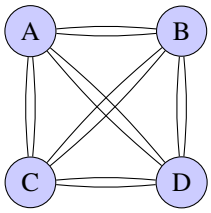
Fig. 2. Introducing a fixed-size buffer to either side of the link, it is possible to maximize the utilization of any wire length.

Your Best Effort Isn't Enough, NACK

Hardware, transmission, and transient failures are ignored in Equation (1) to simplify the model. Composing a probability of packet transmission failure, L, across multiple links for a reliability calculation is a very optimistic view of packet loss on networks. A full-duplex link is a logical grouping of two simplex links that are independent failure domains. Instead of a single L, bidirectional link reliability is represented by two separate probabilities of successful transmission $\langle p_{\leftarrow}, p_{\rightarrow} \rangle, p \in [0, 1]$.

Assuming for a second that a simplex link is either perfectly working or disconnected, a full duplex link is represented by 4 states

- $\langle 0, 0 \rangle$ - Neither direction is connected
- $\langle 1, 0 \rangle$ - One direction is connected
- $\langle 0, 1 \rangle$ - The other direction is connected
- $\langle 1, 1 \rangle$ - Both directions are connected



A fully connected mesh of just 4 nodes has 6 links with 4 possible states creating $6^4 - 1 = 1095$ unique network failures. We subtract one for the case of the fully connected graph.

At the IP layer, large packets may be split into multiple fragments, each containing a portion of the original IP fragment that needs to be reassembled before passing up the stack. If even one fragment is delayed or lost beyond the reassembly timeout, the entire IP datagram is dropped[2].

Let $G = (V, E)$ be a directed graph representing a communication network, where:

- V is the set of nodes.
- $E \subseteq V \times V$ is the set of directed edges (links).

Each directed edge $(u, v) \in E$ is associated with a probability that a sequence of bits is successfully delivered $p_{uv} \in [0, 1]$ and a one-way delay propagation $d_{uv} \in (0, \infty)$.

We define the new edge probability and delay functions:

$$p : E' \to [0, 1], \quad p(u, v) = p_{uv}$$
$$d : E' \to (0, \infty), \quad d(u, v) = d_{uv}$$

To represent a general path W as a walk through the graph, we can define the path as a sequence of vertices, where each pair of consecutive vertices (u_i, u_{i+1}) represents an edge in the graph. The path begins at a source node and ends at a destination node, and it may traverse through several intermediate nodes. On a lossy network, where reliability is recovered by the end-to-end principal, the probability of success is the product of all successful transmission probabilities along the path:

$$P_{\text{success}}(W) = \prod_{i=0}^{n-1} p(u_i, u_{i+1})$$
$$d_{\text{total}}(W) = \sum_{i=0}^{n-1} d(u_i, u_{i+1})$$

[2]By allowing IP datagrams of unbounded size, you are inviting the presence of a L3 switch node that can capture and buffer IP datagrams along the critical path. Switches promise IP datagrams to be stored in the case of burst traffic, but drop packets if their buffers fill up, kicking the burden of reliability one layer higher.

Each IP frame, F, is fragmented into n packets and transmitted on n separate walks $W_0, ...W_{n-1}$, and the ACK packet travels on W_n, so the total probability of success is the product of all probabilities across all hops in every fragment and ACK path.

$$P_{\text{success}}(F) = \prod_{i=0}^{n} P_{\text{success}}(W_i)$$

Expected time for IP frame propagation E_t, with sender timeout T_{timeout} can be calculated as

$$E_t = \max_W d_{\text{total}}(W) + d_{\text{total}}(W_n) + \frac{T_{\text{timeout}}}{1 - P_{\text{success}}(F)}$$

An IP frame fragment on an ACK/NAK ethernet layer continues to operate through transmission failures. From the Bartlett conjecture [7] it is possible to construct a pair of automata that continues to operate through any positive integer i consecutive transmission failures. We present the reliability model for a ACK/NAK network as two separate calculations – the expected time for the transmission of F across a reliable network, and the probability P_{failure} that any fragment or TCP-ACK experiences i consecutive failures that results in retransmission.

To represent the ACK/NAK layer, we define an undirected graph $G' = (V, E')$, where:

$$E' = \{\{u, v\} \subseteq V \mid (u, v) \in E \text{ and } (v, u) \in E\}$$

We define the edge probability and delay functions:

$$p' : E' \to [0, 1], \quad p'(u, v) = p(u, v) \cdot p(v, u)$$
$$d' : E' \to (0, \infty), \quad d'(u, v) = \frac{d(u, v) + d(v, u)}{p(u, v) \cdot p(v, u)}$$

A link-level ACK/NAK scheme can model expected time to successful transmission over a link with two-way latency T_l as one-way propagation delay d.

$$d = \frac{d_{\leftarrow} + d_{\rightarrow}}{p_{\leftarrow} p_{\rightarrow}} = \frac{d_{\leftrightarrow}}{p_{\leftrightarrow}}$$

The expected time to transmission, E_t of a fragment on a reliable path W can be expressed as

$$d'_{\text{total}}(W) = \sum_{i=0}^{n-1} d'(u_i, u_{i+1})$$

An IP frame, F, transmitting on n walks with an ACK is $\{W_0..W_n\}$. The expected time for successful transmission of F assuming $< i$ failures on each link can be approximated as

$$E_t = \max_W d'_{\text{total}}(W) + d'_{\text{total}}(W_n)$$

The state machines are only truly broken if i consecutive failures occur, in which case the packet is

dropped[3]. The expected # of transmissions to witness i consecutive failures with link reliability $p_{\leftrightarrow}$ is calculated using the Markov chain approach and written as a python program $f(i, p_{\leftrightarrow})$ in Appendix A. For reference, 10 consecutive failures on a link of $p_{\leftrightarrow} = 99\%$ is $f(10, 0.99) = 1.8 \times 10^{20}$.

The probability of successfully transmitting before i consecutive failures on an edge (u,v) is

$$q'(u,v) = 1 - p'(u,v)$$

$$P'_{\text{success}}(u,v) = 1 - \left(\frac{q'(u,v)}{1 - p'(u,v) \cdot q'(u,v)^{i-1}} \right)^i$$

$$P'_{\text{success}}(W) = \prod_{i=0}^{n} P'_{\text{success}}(u,v)$$

$$P'_{\text{success}}(F) = \prod_{i=0}^{n} P'(W_i)$$

On a fragment path of 3 links where $\langle p_{\leftarrow}, p_{\rightarrow} \rangle = \langle 0.97, 0.97 \rangle$, a best-effort network will experience a packet failure requiring a retransmission timeout of 200 ms - 1 s every 37,000 frames. An ACK/NAK protocol capable of surviving 5 consecutive failures will experience a packet failure on the same links every 13,000,000 frames. With $d_{\text{total}}(W) << T_{\text{timeout}}$, a 1000x reduction in timeouts a 1000x gain in throughput, and a massive reduction in tail latency.

Reliability up the Stack

The final question is not whether reliable delivery is possible, but whether it is worth it—especially at Layer 2, where guarantees come at the cost of increased complexity. Critics argue that distributed system failures do not stem from transient packet loss, but from deeper causes: node crashes, software bugs, and network partitions that persist longer than retransmission windows. Pat Helland [3] captures this well, noting that the hardest problems in distributed computing often have little to do with packet drops, and everything to do with how systems handle ambiguity.

Yet there is a compelling argument for building reliability in from the bottom. Afek et al. [11] proved a fundamental limitation: in networks that allow reordering, duplication, delay, or loss, reliable end-to-end channels cannot be constructed without sacrificing throughput and latency. Every retransmission algorithm must trade bandwidth efficiency for delivery guarantees, and in doing so, amplify system complexity. A reliable Layer 2 network sidesteps this bind entirely. It simplifies the logic above it, eliminates the need for per-packet deduplication or reordering, and allows higher-layer protocols to reason about failure in cleaner, more declarative terms.

Viewed this way, our model—a train of fixed-size packets acknowledged at every hop by reversible transformations—offers more than just performance. It provides a substrate upon which ACID-like semantics can emerge naturally. To higher layers, the effect is indistinguishable from a transactional channel: every packet is either fully delivered and acknowledged, or not delivered at all. There are no ghosts, no half-completed messages. This clean behavior reduces the mental overhead of building fault-tolerant systems, echoing Lynch's observation that the real complexity in distributed computing lies not in acknowledgments, but in reasoning about state and concurrency.

Larry Peterson et al. [12] observed long ago that the internet lacks a native request/response protocol. TCP masquerades as reliable, but its model of loosely coupled, temporally unconnected packets provides no common knowledge between sender and receiver. Without request/reply message pairs (RRMPs), no formal signaling can occur—a principle well understood in physics. Even in quantum mechanics, non-local correlations do not violate causality. Signaling still demands a light-speed exchange. Spacetime, it turns out, has a handshake.

Where we depart from conventional wisdom is in our skepticism of the end-to-end (E2E) principle as implemented by modern cloud vendors. They argue that endpoint intelligence—combined with a dumb, stateless network—can recover from any failure. 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. The result is a system where packets are dropped, reordered, delayed, or duplicated—exactly the conditions Afek et al. warned about.

This "fire-and-forget" style of communication might be good enough for bulk data transport. But it undermines the foundations of distributed computing. What developers truly need are messages that arrive reliably, in order, and in context. They need protocols that honor causality, not just throughput.

The belief that upper layers can always compensate for lower-layer faults is more than optimistic—it's structurally flawed. Once a message is silently dropped or delayed beyond the visibility horizon of a protocol, the sending event becomes epistemically inaccessible to the receiver. That is, the receiver cannot determine whether the event ever occurred at all. Attempts to rebuild reliability from above amount to guesswork, requiring elaborate retries, heuristics, or consensus algorithms that introduce more complexity and still cannot guarantee correctness. In a distributed system, this epistemic uncertainty doesn't just slow things down—it breaks the

[3]The failure of the state machines are events experienced by both sides of the link. The sending node can use this event to inform the frame owner to retransmit rather than suffer the full sender timeout.

illusion of atomicity, leading to race conditions, inconsistent views of shared state, and irreparable application-level anomalies.

What's needed is not just better abstractions over an unreliable network, but a better substrate. One that aligns information flow with physical causality—where delivery is deterministic, and acknowledgment is not a timeout but a transformation. Only then can higher layers reason reliably about events, without resorting to speculative execution or fragile compensation logic.

Conclusion

Robert Garner reminds us that Ethernet was never intended to be reliable—it was designed to be fast and simple, deferring guarantees to higher layers. And in many contexts, that remains a defensible choice. But as our systems have grown more interconnected, more transactional, and more distributed, the cost of assuming unreliability at the foundation has grown too.

This kind of reliability at Layer 2 is more than a convenience. It is a structural improvement with consequences that ripple upward through the entire stack. When the link itself can be trusted, transport protocols can shed complexity, applications can respond with greater immediacy, and systems can be built on a more predictable substrate. Reliable link-layer protocols are not merely faster. They're structurally superior. They collapse error-handling closer to the cause. They reduce the surface area of failure. And they grant higher layers a cleaner, more deterministic abstraction on which to build.

Garner was right to say that applications need different levels of reliability. But perhaps the deeper insight is this: the closer we place reliability to the wire, the more degrees of freedom we restore to everything above it. When the link can be trusted, the stack can be simplified, and trust in transactional systems can follow.

> *To ACK or not to ACK?* The better question may be this: Where do we begin to build systems we can believe in?

References

[1] R. M. Metcalfe and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, pp. 395–404, July 1976.

[2] M. Alfatafta, B. Alkhatib, A. Alquraan, and S. Al-Kiswany, "Toward a Generic Fault Tolerance Technique for Partial Network Partitioning," *Usenix*, vol. 14th USENIX Symposium on Operating Systems Design and Implementation, pp. 351–368, 2020.

[3] P. Helland, "Fail-fast Is Failing... Fast! Changes in compute environments are placing pressure on tried-and-true distributed-systems solutions.," *Queue*, vol. 19, pp. Pages 60:5–Pages 60:15, Mar. 2021.

[4] G. Rubino, L. A. Rozema, A. Feix, M. Araújo, J. M. Zeuner, L. M. Procopio, Č. Brukner, and P. Walther, "Experimental Verification of an Indefinite Causal Order," *Science Advances*, vol. 3, p. e1602589, Mar. 2017. arXiv:1608.01683 [quant-ph].

[5] G. Chiribella, G. M. D'Ariano, P. Perinotti, and B. Valiron, "Quantum computations without definite causal structure," *Physical Review A*, vol. 88, p. 022318, Aug. 2013. arXiv:0912.0195 [quant-ph].

[6] W. C. Lynch, "Computer Systems: Reliable full-duplex file transmission over half-duplex telephone line," *Commun. ACM*, vol. 11, pp. 407–410, June 1968.

[7] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," *Commun. ACM*, vol. 12, pp. 260–261, May 1969.

[8] J. Gray, "A transaction model," in *Automata, Languages and Programming* (G. Goos, J. Hartmanis, W. Brauer, P. Brinch Hansen, D. Gries, C. Moler, G. Seegmüller, J. Stoer, N. Wirth, J. Bakker, and J. Leeuwen, eds.), vol. 85, pp. 282–298, Berlin, Heidelberg: Springer Berlin Heidelberg, 1980. Series Title: Lecture Notes in Computer Science.

[9] G. V. Bochmann, "Finite state description of communication protocols," *Computer Networks (1976)*, vol. 2, pp. 361–372, Sept. 1978.

[10] J. M. Wozencraft and M. Horstein, "CODING FOR TWO-WAY CHANNELS,"

[11] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang, and L. Zuck, "Reliable communication over unreliable channels," *J. ACM*, vol. 41, pp. 1267–1297, Nov. 1994.

[12] L. Peterson, N. Hutchinson, S. O'Malley, and M. Abbott, "RPC in the x-Kernel: evaluating new design techniques," *SIGOPS Oper. Syst. Rev.*, vol. 23, pp. 91–101, Nov. 1989.

Appendix

Python Code for Calculating the Expected Number of Trials to State Machine Failure

The function $f(n, p)$ calculates the expected number of trials to get n consecutive failures in a sequence of Bernoulli trials with probability p of transmission failure.

```python
def f(n, p):
    # q = probability of 1
    q = 1 - p

    # Initialize the E array for E_0 to E_n
    E = [0] * (n + 1)

    # E_n = 0 (base case)
    E[n] = 0

    # Fill the E array from E_(n-1) down to E_0
    for k in range(n - 1, -1, -1):
        E[k] = 1 + q * E[k + 1] + p * E[0]

    # Solve for E_0 by iterating again
    for k in range(n - 1, -1, -1):
        E[0] = (1 + q * E[k + 1] + p * E[0])
        E[0] /= (1 - p)

    return E[0]
```