The Case for Reliable Atomic Links

PROBLEM:

Many latency, Inconsistency, and Unbounded problems encountered in distributed systems today arise from our inability to distinguish between a node that is merely slow from one that has failed or become unreachable due to network failure.

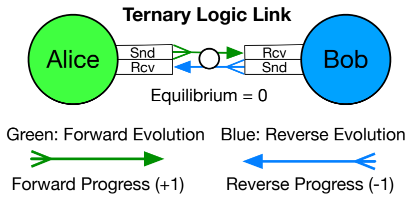

Figure 1: Two CELLs and a LINK with conserved quantities, epistricted with Ternary, or Three-valued logic

We take the two most recognizable elements in datacenters today: *servers* and *switches*, and refactor them into simpler, fully *independent* failure domains: CELLs and LINKs. A CELL is a universal node: an autonomous unit of compute, storage *and* packet processing. A LINK is a bidirectional tunnel-element; an autonomous communication entity between *two* CELLs)[1]. Physically, the LINK comprises the cable and SerDes' on both ends to form a self contained execution environment.

Unifying node elements makes things simpler because we have only one type of node to manage instead of two. We raise the notion of a LINK to *first order* – a first-class citizen in the infrastructure – a bipartite element of conserved information with two *complementary* halves – *persistable* through failure and recovery events. i.e., a communication object that doesn't rule out that some *local* fault-detection and computation is involved.

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[2] 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) [3]. TIKTYKTIK andIUI properties are mathematically *compositional*.

Trees of LINKs provide a deterministic *conserved quantities* mechanism to reliably distribute *indivisible* tokens among *agents* on an application graph. Intermediate CELLs *promise* [4] to never loseIUI tokens. This defends against lost tokens because if any part of the chain (or trtee) breaks. Alternate paths are available to seamlessly recover the conserved quantity and continue operation[5].

[1] Think of CELLs in Cellular automata. Think of LINKs as compute elements with their own autonomous and independent failure domain (Local INformation and Knowledge). In the same way devices within a single execution environment are considered a single failure domain; the LINK can detect device and cable errors and make the machine stop. The challenge is how to get it started again. The solution is triangle networks: Tripartite directly connected graphs.

[2] 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.

[3] 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 forIUI.

[4] The alternative definition of Futures/Promises also applies: execution is primed to create future *liveness events*.

[5] LINKs t provide a more reliable foundation for distributed system *services* for applications: consensus, atomic broadcast, leader election, cluster membership and distributed arithmetic, Which, in-turn, enable higher level functions such as *network-assisted-transactions* which serve, for example, distributable (scale-out), persistable (durable), and lightweight (but-accurate) accounting of *conserved quantities* associated with application state or resource usage.

For maximum efficiency, both sides of the link should be plesiochronous; keeping *snakes* circulating at the same rate through both transmit and receive channels.

A New Law: Conservation of TOKENS

By strengthening the system model, `LINK` s and `IUI` tokens provide a general foundation to solve many distributed-systems problems such as failure detection, consensus, and atomic transactions.

Problem

- Sixteen years ago, CPU-Bound tasks, such as buffer pool and concurrency control, were the primary bottlenecks in OLTP databases. Today, communication overhead is the dominant factor affecting performance*.
- Timeouts & retries (TAR) are the "root of all evil" in distributed systems. They lead to Congestion, Cascade Slowdowns: Retry Storms, Metastable Failures, Limpware, Transaction Failure, and Silent Data Corruption – which leads to lost and corrupted transactions in all distributed systems, including databases.

Failure Modes

One might imagine[6] if each host (or its SmartNIC) maintains its half of the *shared state*, then shouldn't the switched network be able to act as a proxy for a single *logical* LINK? When a switched network fails, and reroutes, can't the two sides (SmartNICs) just stitch the two halves of the *shared state* back together again? we could achieve the properties of `LINK`s over existing switched networks.

This simple hazard analysis[7] misses a fundamental issue: *networks don't maintain state on behalf of applications*. Switches drop packets (and state) whenever *they* feel like it, so there are many more ways for *logical link ts* to get confused over switched networks and compromise the integrity of the *shared state*.[8]

Key issue: Switched networks drop packets anywhere along the network path; eradicating state *and* events needed to maintain *promises* and *liveness* respectively. When a `LINK` fails, both sides are preserved. If there is an atomicity violation in the `IUI` it can always be detected, and retransmissions occur only on a *real* failure (such as disconnection–where alternative routes are explicitly coordinated with applications), thus enforcing that `IUI` tokens have no duplicate or out of order deliveries on the `LINK`.

* OLTP Through the Looking Glass 16 Years Later: Communication is the new Bottleneck

[6] Such a recovery mechanism is *not* available through conventional switched networks; because of the uncertainty of how many packets were lost, exactly where along the path through the switched network they were lost, how many were duplicated, and how they might have been re-ordered in the switched network.

[7] Typical hazards: (1) Servers with a LINK to a single Top of Rack Switch (ToR) are unable to heal at all; there is only one path from the server to *anyone*. (2) ToRs represent SPoFs; when they fail, many servers (in that rack) also become unavailable. Worse still: the servers don't go down, they keep on computing but don't know they are isolated. ToRs have also been known to exhibit complex cascade failures where a firmware failure triggered in one will take down many neighbor TORs.

[8] Even without failures, the behavior of switched networks forces us into a high overhead approach. If packets can be indiscriminately dropped, delayed, duplicated and reordered, we have no choice but to implement `TCP/IP` sender timeouts, Even in optimal situations, TCP introduces high complexity and overhead and unbounded latency for *all* operations, which is too slow. There is no escape: if we drop, delay duplicate or reorder packets, you have to effectively implement TCP. If we can eliminate duplication and re-ordering, recovery is far simpler, and removes the tail latency from normal operations.

Switched Networks

When packets are dropped in a switched network, more than information is lost, *events* are lost, and it becomes extraordinarily difficult to recover both sides of a *shared state* that stretches across even a single switch. [9]

Not sure dropped packets are the right hazard to mention here "the protocol is exquisitely sensitive to packet loss". What really destroys state as shown in the Afek et al. paper was RACE CONDITIONS on every packet transmission (delay + reorder), with the proposed solution being essentially E2E Stop and wait

A *directly connected* `LINK` can *promise* three things a switched network cannot: (a) maintain an ordering of events (heal, send packets, fail, heal again, send more packets) – essential for non-idempotent and non-commutative operations. (b) not drop *certain* packets without notification – essential for recovery. And (c) maintain complementary *direction* state – essential for distributed self-stabilizing algorithms with local rerouting rules for reliable trees. [10] [11]

What's necessary is an *entanglement* between state machines – locking them together silently in normal operation, and failing locally at 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 Trade-offs]. To put it in terms an engineer can internalize, a system that fails instantly, can heal immediately.

Bipartite Integrity and the E2E Principle

The End-to-End (E2E Principle [12]

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. The LINK combines the failure domains of SerDes, cables, connectors and atomic rays?? (Idea is no matter what happens, the link freezes If anything is amiss, you can remove if flow is better – into one failure hazard `LINK`s are independent failure domains, with (effectively) one failure hazard: *disconnection*[13]; which is straightforward to recover from. Switched networks, on the other hand, have many more failure hazards: they indiscriminately *The Network is Reliable An informal survey of real-world communications failures* drop, delay, duplicate and reorder packets – that's just the way networks behave – justified by the end to end argument

[9] LINKs do not reorder or duplicate packets so we can now use a high-performance `payload operator in Atomic Ethernet` for coherence; only paying the performance cost of `TCP` when it fails for real, i.e. disconnection. Remarkably, this also paves the way for very high bandwidth utilization for datacenter to datacenter application flows, because it eliminates the most difficult aspects of reordering and duplication of packets. In scale-out/massively distributed architecture.

[10] Actually, a `LINK` can promise many more than just these three things: whatever property the agent or application wishes to attach to the bipartite `LINK` object.

[11] Reversible`IUI` enables the reversal of non-idempotent structures in distributed applications.

[12] The end-to-end principle states that in a general-purpose network, application-specific functions ought to reside in the end hosts of a network rather than in intermediary nodes, provided that they can be implemented "completely and correctly" in the end hosts. Our claim is that it is impossible to implement *bidirectional* synchronization primitives "completely and correctly" without`IUI` (or something very much like it). The datacenter is not the Internet, and Saltzer, Reed, & Clark considered one-way (unidirectional) *file transfer*, not the *bidirectional* synchronization of replicas (token coherency) described here.

[13] 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.

Reversibility

In physics, time-symmetry is the universal property by which energy is conserved. In OAE, reversibility ensures the conservation ofIUI tokens because traffic on the link stops, and the complimentary state on both sides of the link is preserved until higher level protocols (triangle relationships) where the 3rd CELL (Transaction Manager Role) is able to perform "successive reversibility", heal the link, and return the protocol back to the equilibrium state.

The shared state TIKTYKTIK property can also be used to mitigate broadcast storms in network rendezvous, timeout storms in microservices, or reconstruction storms in erasure coded storage[14]. InIUI, packets are not merely dropped, they are replaced with special events denoting failure, to maintain *liveness*. Because LINK failures are *independent* (from node failures) we can successively recover individual disconnection failures.

This single step recovery mechanism paves the way forIUI to reverse *one or more* steps in distributed systems which use non-idempotent or non-commutative data structures. [15].

Disconnection is the Most Likely Failure Hazard in LINKs

Packets delayed by disconnected LINKs don't threaten *liveness* or the integrity of the *shared state*. Switched network hazards include: indiscriminately *dropped*, *delayed*, *duplicated* and *reordered* packets. Conventional mitigations (e.g. TCP) add significant complexity and performance overheads, and still fail to solve the problem.

Examples

The advantage of the *shared state* is that both sides *know* the LINK is broken which can't be done through a switched network with *even a single switch* in series. LINKs simplify some important distributed system algorithms such as consensus ,two-phase commit, and reliable tree generation:

Paxos [16] "Agents operate at arbitrary speed, may fail by stopping, and may restart. Since all agents may fail after a value is chosen and then restart, a solution is impossible unless some information can be remembered by an agent that has failed and restarted". [17] The assumption is when a node has failed and restarted, it can't remember the state it needs to recover. WithIUI, the other half of the LINK can tell it the state to recover from. This avoids the performance cost of heavyweight transactions and persistent storage.

[15] The *shared state* can be compromised by duplicated or reordered packets, but it is resilient to lost or delayed packets.

[14] LINKs also coalesce heartbeats to make microservices more scalable, and make failure detectors *reliable*.

[16] Paxos is mentioned first because it is a clearer example how reliable L2 impacts the application consistency tradeoffs. Reliable Paxos probably could be its own paper.

[17] Prepare phase is not about bad networking, but to coordinate between competing proposals. It cannot be eliminated, but its performance can be improved by no longer relying on timeouts.

Two-phase commit The prepare phase is asking if the receiving agent is ready to accept the token. This serves two purposes: communication liveness and agent readiness. LINKs[18] provide the communication liveness test, and we can avoid blocking on agent ready, by having the LINK store the token on the receiving half of the LINK. If there is a failure, both sides (senders and receivers in both NICs) know; and both sides know what to do next.

Why can't the sending side on Paxos keep the information the receiving side needs in case it fails? The other side of the LINK knows: the state that was lost in the crash is maintained by the half of the LINK on the other side. In leader election a master sends a request to a worker (prepare) and it doesn't get the ack. With IUI, the master knows definitively, if it is a neighbor.

In two-phase commit (2PC) there is no safety proof. In three-phase commit, there is no liveness proof. Can we also often eliminate the prepare phase in consensus and 2PC? [19] which improves performance.

One or more corner cases are eliminated where the replicated state machine can get blocked due to loss of acknowledgement .

Atomicity

Atomicity in IUI provides significant advantages in distributed computing by making corner cases disappear, it makes it *simpler* to implement those distributed algorithms correctly, and the number of circumstances where things go wrong is reduced by (potentially) several orders of magnitude.

Reliable tree generation Binary LINK reversal algorithms[20] work by reversing the directions of some edges. Transforming an arbitrary directed acyclic input graph into an output graph with *at least one route from each node to a special destination node*. The resulting graph can thus be used to route messages in a loop-free manner[21]. LINKs store the direction of the arrow (head and tail);IUI facilitates the atomic swap of the arrow's tail and head to maintain loop-free routes during failure and recovery.

Those examples are applications using multiple links composed together, as depicted in our Atomic Ethernet logo.

Also, liveness becomes trivial in paxos. As long as a majority is up, progress is guaranteed. The real problem is cluster membership, when new nodes join, or when old nodes leave, either voluntarily or involuntarily.

[18] LINKs exploit a unique combination of physics, electrical engineering and computer science. Think ofIUI as the distributed systems (network- based) equivalent of the atomic Compare And Swap (CAS or just plain SWAP) primitive used in shared memory architectures. The result is handshake-free messaging with strong liveness, recoverability and security properties.

[19] And in three-phase commit, the pre-prepare phase?

[20] Charron-Bost et. al. generalize the Gafni-Bertsakas (GB) binary LINK reversal algorithm.

[21] LINK reversal algorithms don't generate shortest paths, just some paths. However, they do generate *multiple* loop-free routes. This allows the LINK to inform the agent (and the application, if needed) when switching to an alternate failover tree. As long as pre-computed failover paths are available, we used the Dynamic Tree Algorithm (DTA). Only when CELLs lose *all* their paths do they need to participate in the LINK-reversal algorithms, which means less communication overhead and more stability.

FAQ

[QUESTION I still do not understand how two devices connected using a full duplex Ethernet link can support reliable communications without time outs and retries

[ANSWER] You are right at a conceptual level, but the reality of timeouts and retries is so different with a Stop-and-Wait (SaW) protocol as to make you wrong. Let us explain.

- In current networks, timeouts and retries are end to end concepts. Alice sends an application message to Bob with TCP, which attempts to guarantee delivery by sending ACKs. Those ACKs can be lost on the way from Bob's node to Alice's. Alice has no way to know if part of the message or the ACK was lost or delayed, so she retries after a timeout.
- With Stop and Wait (SaW) (the alternating bit protocol) , reliability is at the link level, where the protocol provides sufficient common knowledge to address the majority of the issues. In particular, if the link doesn't break, the packets get through. There's no need to drop a packet due to a full buffer, because the SaW signal is credit based flow control.
- If the link breaks noisily (both sides get a signal), both sides know which phase of the protocol they are in, so they know which one is responsible for forwarding the packet. I think you'd be stretching the concept if you said A sending the packet on the new path is a TCP-like retry,

If the link breaks silently or even in only one direction, both sides know that a SaW signal hasn't arrived. Since both sides know that both sides know (common knowledge), each can declare the link dead with full knowledge that the other side will eventually declare it dead too. Of course, knowing when to stop waiting for the signal is the moral equivalent of a timeout, but there's no need to coordinate on a value. I'd say that's quite different from the timeouts you're thinking of.

End Notes:

18. Charron-Bost et. al. generalize the Gafni-Bertsakas (GB) binary LINK reversal algorithm. Which works by reversing the directions of some edges. Transforming an arbitrary directed acyclic input graph into an output graph with *at least one route from each node to a special destination node*. Charron-Bost et. al.

19. Link reversal algorithms don't generate shortest paths, just some paths. However, they do generate *multiple* loop-free routes. This allows the LINK to inform the agent (and the application, if needed) when switching to an alternate failover tree. As long as precomputed failover paths are available, we used the Dynamic Tree Algorithm (DTA). Only when CELLs lose *all* their paths do they need to participate in the LINK-reversal algorithms, which means less communication overhead and more stability.

20. The shared state integrity of the single physical LINK is a *promise* that two NIC's can make with each other only over a single physical LINK. This is explicitly an *anti-promise* for conventional switched networks.

Conclusion

Æthernet LINKs require a direct physical connection; their benefits cannot be achieved over switched networks composed of a chain of unreliable links. Atomicity + Self-stabilizing algorithms, provide a general foundation to solve many distributed systems problems, and mitigate broadcast, timeout and reconstruction storms in networks and distributed storage.

[22]

[22] The shared state integrity of the single physical LINK is a *promise* that two NIC's can make with each other only over a single physical LINK. This is explicitly an *anti-promise* for conventional switched networks.