Dispatch No. 010 · Reachability, Routing and the P in CAP, Part 1

Pick Any Two, Revisited

The CAP theorem from the network side of the wire.

M.J. Martin
Outlan.net
November 2026 · Version 1.1.2

ABSTRACT

The CAP theorem is usually taught as a database result: during a network partition, a replicated system can stay consistent or stay available, but not both. The P in CAP, however, is a property of the network, and network engineers build, operate and change the thing that partitions. This article revisits the theorem through its two-server proof, then looks at what a partition is from the network's point of view (a network failure that pushes communication between replicas past a timeout, as distinct from discovery and application failures that only look like one) and at the CAP choices that routing protocols, DNS, high-availability pairs, DHCP failover and policy controllers already make. It then covers the trade that applies when nothing is broken (latency versus consistency, or PACELC) and closes with the organizational reason these choices are so rarely written down: in most enterprises, no single team owns the timeout where the choice is made.

1. INTRODUCTION

About a decade ago I put a slide titled "Pick any two – The 'CAP Theorem'" into an architecture deck. The slide did what that phrasing always does. It made a hard result easy to remember and easy to misapply. An illustrated proof of the theorem that has circulated for years [1] is a good excuse to take another look, this time from the network side of the wire.

Most treatments of CAP are written by and for database people. That makes sense, since the theorem is about replicated data. The P in CAP (partition tolerance), however, is about what happens when the network stops carrying messages between replicas. The network is ours. We build it, we change it on Tuesday nights, and we own the timers that decide how long a failure lasts before anything notices. The systems we run (routing protocols, DNS, firewall pairs, SD-WAN controllers and so on) also make CAP choices of their own, every day, whether anyone chose them or not.

In this article, we first look at where the theorem came from and what it actually proves. Next, we look at what a partition is from the network's point of view, and at the CAP choices that common network systems already make. Then we look at the trade that applies when nothing is broken. We close with the question this newsletter keeps coming back to: why is it so hard to just do the right thing?

2. A SHORT HISTORY

Eric Brewer stated the idea as a conjecture in a keynote at the ACM Symposium on Principles of Distributed Computing (PODC) in 2000 [2]. Two years later, Seth Gilbert and Nancy Lynch published a formal proof for the asynchronous network model [3]. The two decades since have mostly been spent arguing about what the three letters mean. Table 1 lists the milestones worth knowing.

Table 1. A short history of the CAP theorem

Year	Work	Contribution
2000	Brewer, PODC keynote [2]	States the conjecture: a web service cannot guarantee consistency, availability and partition tolerance at once.
2002	Gilbert and Lynch [3]	Proves it. Defines consistency as atomic (linearizable) reads and writes, and availability as a response to every request at every non-failing node.
2010	Hale [4]	Argues that partition tolerance is not optional in a distributed system, so the real choice is C or A while a partition lasts.
2012	Brewer [5]	Calls "2 of 3" misleading. Partitions are rare, the choice can differ by operation, and the hard work is detecting a partition and recovering from it.
2012	Abadi [6]	Proposes PACELC: if partitioned, trade availability against consistency, else trade latency against consistency.
2014	Bailis and Kingsbury [7]	Survey real-world network failures and conclude that partitions are a routine operational fact, not a theoretical edge case.
2015	Kleppmann [8]	Asks the industry to stop labeling databases "CP" or "AP" and to describe specific guarantees instead.
2017	Brewer [9]	Argues that Google Spanner is technically CP but effectively CA, because partitions on Google's network are rare enough to ignore for most users.

3. THE PROOF IN FIVE STEPS

The proof is short enough to carry in your head, which is part of its appeal. It uses three narrow definitions [3]:

- **Consistency** – A read that starts after a write has completed returns that value or a newer one, no matter which server answers. Formally, this is linearizability (every operation appears to take effect at a single instant).
- **Availability** – Every request received by a non-failing server gets a response. Refusing to answer, or waiting forever, does not count.
- **Partition tolerance** – The system keeps its guarantees even when the network drops some or all messages between servers.

An everyday analogy helps. Picture two bank branches that keep one shared ledger by calling each other after every transaction. The phone line goes dead. A customer deposits $100 at the first branch. A second customer (or the same one, five minutes later) asks the second branch for the balance. The second branch has two choices. It can answer with the balance it last heard about, which is wrong (available, not consistent). It can tell the customer to come back when the phones work (consistent, not available). There is no third answer. The formal proof [1][3] follows the same path with two servers, G1 and G2, that each hold a copy of one variable:

1. Assume a system exists that is consistent, available and partition tolerant.
2. Partition the network so G1 and G2 cannot exchange any messages.
3. A client writes a new value to G1. Availability requires G1 to acknowledge the write. The partition stops G1 from telling G2.

4. A second client reads from G2. Availability requires G2 to answer, and the only value it has is the old one.
5. The client reads the old value after the new value's write was acknowledged. That violates consistency, so the system assumed in step 1 cannot exist.

> **NOTE** The theorem is narrower than the slogan. It rules out perfect consistency combined with perfect availability while a partition lasts. It says nothing about the system when the network is healthy, and nothing about weaker consistency models (causal ordering, read-your-own-writes and so on) that can stay available during a partition. Brewer made this point himself in 2012 [5]. "Pick any two" also implies that a distributed system can choose to give up P. It cannot. The network decides when a partition happens, and the system only decides how to behave during one [4].

4. THE P BELONGS TO THE NETWORK

Brewer's 2012 paper contains the sentence that matters most to network engineers. In practice, he writes, a partition is a time bound on communication [5]. A node that cannot reach its peer within its timeout must either cancel the operation (giving up availability) or proceed without the peer (risking consistency). Retrying only delays the decision.

By that definition, a cut fiber and a path running at 30 percent packet loss look the same to an application once the timer fires. The list of things that count as a partition gets much longer than "the WAN link went down":

- **Hard failures** – A fiber cut, a failed line card or a power event in one site. The textbook case, and in most networks the least common one.
- **Lossy or congested paths** – Loss and queueing push round-trip time past the application's timeout, while the routing protocol still reports the link as up.
- **Routing convergence** – A BGP withdrawal or an IGP reconvergence that takes seconds to settle. Brief, but long enough to trip a replication timeout set in milliseconds.
- **MTU black holes** – Small packets (heartbeats, keepalives) pass and large packets (replication traffic) are dropped. Health checks report a healthy peer while replication stalls, which is a partial partition that is hard to see.
- **One-way drops** – A stateful firewall or an asymmetric route drops return traffic in one direction only. One side believes the peer is alive, the other believes it is gone. Brewer notes that one-sided partitions like this are possible and that nodes can disagree about whether a partition exists [5].

Not every failure that looks like a partition is one. On October 19 and 20, 2025, Amazon Web Services lost DynamoDB in its Northern Virginia (us-east-1) Region for close to three hours of primary impact [10]. The root cause was a latent race condition in DynamoDB's automated DNS management, which left an empty DNS record for the regional endpoint. Every client using the public endpoint, including other AWS services, failed to connect, and recovery was gated by DNS caching: clients reconnected as their cached records expired, between 2:25 and 2:40 AM PDT [10]. To those clients the outage looked exactly like a partition, because the connection never completed within its timeout. The network, however, was healthy throughout. The service's addresses were reachable, and a client holding a cached or fixed address could have connected. What failed was discovery (the mapping from a name to an address), not the network's ability to carry messages.

The distinction matters, because applications stop working for many reasons that have nothing to do with the network: a bad deploy, an expired certificate, an identity provider outage, resource exhaustion, a failed dependency several layers back and so on. They all present the same symptom, a timeout, and the network is usually the first suspect because it is the most visible shared dependency. When the network is the cause, it is usually a change rather than the technology. Uptime Institute's seventh annual outage analysis attributes 30 percent of end-to-end IT service outages to networking and connectivity, and half of the major network-related outages it tracked cite configuration or change management failure as a cause [13]. CAP, as used in this article, is scoped to the narrower case: the network failing to carry messages between replicas.

"The network is reliable" is the first of the eight fallacies of distributed computing attributed to Peter Deutsch [11]. Bailis and Kingsbury collected real failures from cloud providers, enterprise data centers and application operators, and found partitions in all of them [7]. Where application teams tend to treat a partition as a rare event, network teams see its causes in routine change windows (a maintenance reroute, a firewall policy push, a DNS migration).

5. NETWORKS ALREADY CHOOSE

Network systems are distributed systems, and most of them made their CAP choice long before anyone drew the triangle. Table 2 lists the common ones. The "lean" column is a shorthand, with Kleppmann's warning in mind [8]: each entry describes behavior during a partition, not a label for the product.

Table 2. CAP choices that common network systems already make

System	Behavior during a partition	Lean	What it costs
Routing protocols (BGP, OSPF, IS-IS)	Each router keeps forwarding on its own view of the topology until it converges.	Available	Transient loops and black holes ("micro-loops") during convergence.
DNS with caching resolvers	Resolvers keep serving cached answers until the TTL (time to live) expires.	Available	Stale answers, or bad ones, persist until caches age out [10].
Active/passive firewall or load balancer pair	If the heartbeat path fails, the passive unit can promote itself.	Available, by default	Two active units with the same addresses ("split brain") and dropped asymmetric sessions.
MLAG pair (for example, Cisco vPC)	If the peer link fails while the keepalive still works, the secondary suspends its member ports.	Consistent	Half the downstream capacity goes away to keep one forwarding brain.
DHCP failover pair	When the partners lose contact, each server allocates only from its own share of free addresses.	Both, by splitting the data	The usable free pool is cut in half until an operator or timer declares the partner down.
SD-WAN or SASE controller	In a common design, edges keep forwarding on last-known policy, and new policy waits for the controller.	Data plane available, control plane consistent	Policy changes, and revocations, stall until the partition heals.
ZTNA or NAC decision point	Fail-open admits on cached or default policy. Fail-closed denies.	Open: available. Closed: consistent.	Security exposure on one side, an outage on the other.

Three rows deserve a closer look. One shows the problem, one shows the fix, and one shows where the choice gets uncomfortable.

The firewall pair is the network's version of the two-server proof. The heartbeat link is the message path between G1 and G2. When it fails, each unit has to decide on its own whether its partner is dead or merely unreachable. A unit that promotes itself chooses availability, and if the partner is still alive the result is two active firewalls fighting over the same addresses. The fix follows straight from the proof: give the pair a way to break the tie. A second heartbeat path on separate hardware, or a third-party witness, lets a partitioned unit learn that it is in the minority and stay passive. That is choosing consistency, deliberately.

DHCP failover is the model worth copying. The protocol splits the free address pool between the two servers ahead of time. When the partners lose contact, each one keeps serving clients from its own share, so both stay available and neither can hand out an address the other already leased. The pool is the data, and the design sidesteps the C-or-A choice by partitioning the data before the network partitions it. When the outage drags on, an operator (or a configured timer) declares the partner down, and the survivor takes the whole pool. That is every element Brewer asked for in 2012 [5]: an explicit partition mode, limits on what each side may do while partitioned, and a defined recovery.

The security rows are where the choice gets uncomfortable. A ZTNA policy decision point that cannot reach its identity source has to admit or deny. As a rule, the right failure mode for a security control is closed, but not necessarily always. A badge reader that fails closed during a fire alarm is a different problem from a VPN concentrator that fails closed during a regional outage. The point is that someone should make that call on purpose and write it down.

6. WHEN NOTHING IS BROKEN: PACELC

CAP only describes behavior during a partition. Daniel Abadi pointed out in 2012 that the trade most systems pay every day is a different one [6]. His formulation, PACELC (pronounced "pass-elk"), reads: if there is a partition (P), trade availability (A) against consistency (C), else (E) trade latency (L) against consistency (C).

The latency half is pure physics, and it is also ours. Light in fiber travels at roughly 200,000 kilometers per second, or about 5 microseconds per kilometer. New York to London is about 5,600 kilometers on a great circle, so the best possible round trip is about 56 milliseconds before any cable detour or queueing. A replicated write that must be acknowledged by a quorum in another region pays at least one of those round trips, every time. A system that answers from the local replica without waiting pays nothing in latency and accepts that the remote replica is briefly behind. Neither choice is wrong. Both depend on a number the network team controls and the application team rarely asks about.

Brewer's 2017 paper on Spanner makes the same point from the other direction [9]. He argues that Spanner is technically a CP system, because it chooses consistency when a partition occurs, but that users can treat it as CA because partitions on Google's private network are rare. Read closely, much of the case for Spanner's availability is a case for the network underneath it: redundant paths, tight operational control and a lot of engineering spent keeping partitions rare. The theorem did not change. The probability of P did.

7. WHY IS IT SO HARD TO JUST DO THE RIGHT THING?

If the C-or-A choice is made at a timeout, the next question is who owns the timeout. In most enterprises the honest answer is nobody, or rather everybody, separately. The application team inherits a client library's default request timeout. The database team sets a replication timeout. The network team tunes BFD (Bidirectional Forwarding Detection) and routing hold timers. The security team sets the firewall pair's heartbeat interval. Each value is reasonable on its own. Nobody puts them on one page, so the system's partition behavior is not designed at all. It emerges, and it is usually discovered during the first real incident.

Measurement makes it worse. Network availability is reported per device and per link ("five nines on the core"). The availability that matters to a replicated system is per conversation: can replica A reach replica B within its timeout, right now? A core with no device outages can still deliver a partition, through convergence, loss or a one-way drop, and none of it shows up in the availability report.

None of this needs new technology. It needs a few habits:

- **Write down, per operation, which side wins.** Brewer's own advice is that the choice can differ by operation [5]. Payment writes can stop during a partition while catalog reads keep serving from the local copy. Make that a design decision, not a library default.
- **Put the timers on one page.** Line up BFD and routing hold timers, HA heartbeat intervals, replication timeouts and application request timeouts. If the application gives up before the network reconverges, the application decided what a partition means, and it is a good idea to know that before the outage rather than after.
- **Test the partitions you actually get.** Pulling a cable tests the easy case. Inject loss, one-way drops, MTU black holes and DNS failures, and watch what the system does. Kingsbury's Jepsen analyses showed many distributed databases failing these tests in ways their documentation did not mention [12].
- **Give every pair a tiebreaker.** A second heartbeat path, a witness or a quorum of three. Two nodes with one link between them is the exact setup the proof uses.
- **Document, document, document.** The partition-mode behavior of each system belongs in its runbook, next to the recovery steps. Recovery is where Brewer says the hard work lives [5], and it goes much better when someone wrote it down in advance.

8. CONCLUSION

Kleppmann is right that "CP" and "AP" are too coarse to describe real systems [8]. The theorem's question still earns its slide: when the network stops carrying messages, which promise does this system break? For network engineers, the question has a second half. Which of our devices are making that choice right now, and who chose it? The proof needs only two servers and one dropped message. Most production networks supply both on a routine basis.

REFERENCES

[1] "An Illustrated Proof of the CAP Theorem," mwhittaker.github.io, https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/.

[2] E. Brewer, "Towards Robust Distributed Systems," keynote, *ACM Symposium on Principles of Distributed Computing (PODC)*, July 2000.

[3] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[4] C. Hale, "You Can't Sacrifice Partition Tolerance," codahale.com, October 2010.

[5] E. Brewer, "CAP Twelve Years Later: How the 'Rules' Have Changed," *IEEE Computer*, vol. 45, no. 2, pp. 23–29, February 2012.

[6] D. J. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP Is Only Part of the Story," *IEEE Computer*, vol. 45, no. 2, pp. 37–42, February 2012.

[7] P. Bailis and K. Kingsbury, "The Network Is Reliable: An Informal Survey of Real-World Communications Failures," *ACM Queue*, vol. 12, no. 7, 2014.

[8] M. Kleppmann, "Please Stop Calling Databases CP or AP," martin.kleppmann.com, May 11, 2015.

[9] E. Brewer, "Spanner, TrueTime and the CAP Theorem," Google, February 2017.

[10] Amazon Web Services, "Summary of the Amazon DynamoDB Service Disruption in the Northern Virginia (US-EAST-1) Region," https://aws.amazon.com/message/101925/, October 2025.

[11] P. Deutsch, "The Eight Fallacies of Distributed Computing," Sun Microsystems, c. 1994.

[12] K. Kingsbury, Jepsen distributed systems safety analyses, https://jepsen.io/analyses.

[13] Uptime Institute, 7th Annual Outage Analysis, as reported in *Network World*, May 2025, https://www.networkworld.com/article/3980366/networking-errors-pose-threat-to-data-center-reliability.html.