DOI:10.1145/2643130

Article development led by acmqueue
queue.acm.org

An informal survey of real-world communications failures.

BY PETER BAILIS AND KYLE KINGSBURY

The Network Is Reliable

"THE NETWORK IS RELIABLE" tops Peter Deutsch's classic list of "Eight fallacies of distributed computing," all [of which] "prove to be false in the long run and all [of which] cause big trouble and painful learning experiences" (https://blogs.oracle.com/jag/resource/Fallacies.html). Accounting for and understanding the implications of network behavior is key to designing robust distributed programs—in fact, six of Deutsch's "fallacies" directly pertain to limitations on networked communications. This should be unsurprising: the ability (and often requirement) to communicate over a shared channel is a defining characteristic of distributed programs, and many of the key results in the field pertain to the possibility and impossibility of performing distributed computations under particular sets of network conditions.

For example, the celebrated FLP impossibility result[9] demonstrates the inability to guarantee consensus in an asynchronous network (that is, one facing indefinite communication *partitions* between processes) with one faulty process. This means that, in the presence of unreliable (untimely) message delivery, basic operations such as modifying the set of machines in a cluster (that is, maintaining group membership, as systems such as Zookeeper are tasked with today) are not guaranteed to complete in the event of both network asynchrony and individual server failures. Related results describe the inability to guarantee the progress of serializable transactions,[7] linearizable reads/writes,[11] and a variety of useful, programmer-friendly guarantees under adverse conditions.[3] The implications of these results are not simply academic: these impossibility results have motivated a proliferation of systems and designs offering a range of alternative guarantees in the event of network failures.[5] However, under a friendlier, more reliable network that guarantees timely message delivery, FLP and many of these related results no longer hold:[8] by making stronger guarantees about network behavior, we can circumvent the programmability implications of these impossibility proofs.

Therefore, the degree of reliability in deployment environments is critical in robust systems design and directly determines the kinds of operations that systems can reliably perform without waiting. Unfortunately, the degree to which networks are *actually* reliable in the real world is the subject of considerable and evolving debate. Some have claimed that networks are reliable (or that partitions are rare enough in practice) and that we are too concerned with designing for theoretical failure modes. Conversely, others attest that partitions do occur in their deployments, and that, as James Hamilton

IMAGE BY DALIBOR ZIVOTIC

of Amazon Web Services neatly summarizes "network partitions should be rare but net gear continues to cause more issues than it should" (http://bit.ly/1mD8E3q). So who's right?

A key challenge in this discussion is the lack of evidence. We have few normalized bases for comparing network and application reliability—and even less data. We can track link availability and estimate packet loss, but understanding the end-to-end effect on applications is more difficult. The scant evidence we have is difficult to generalize: it is often deployment-specific and closely tied to particular vendors, topologies, and application designs. Worse, even when organizations have a clear picture of their network's behavior, they rarely share specifics. Finally, distributed systems are designed to resist failure, which means *noticeable* outages often depend on complex interactions of failure modes. Many applications silently degrade when the network fails, and resulting problems may not be understood for some time, if ever.

As a result, much of what we believe about the failure modes of real-world distributed systems is founded on guesswork and rumor. Sysadmins and developers will swap stories over beer, but detailed, public postmortems and comprehensive surveys of network availability are few and far between. In this article, we'd like to informally bring a few of these stories (which, in most cases, are unabashedly anecdotal) together. Our focus is on descriptions of actual network behavior when possible, and (more often), when not, on the implications of network failures and asynchrony for real-world systems deployments. We believe this is a first step toward a more open and honest discussion of real-world partition behavior, and, ultimately, toward more robust distributed systems design.

Rumblings From Large Deployments

To start off, let's consider evidence from big players in distributed systems: companies running globally distributed infrastructure with hundreds of thousands of servers. These

reports perhaps best summarize operations in the large, distilling the experience of operating what are likely the biggest distributed systems ever deployed. These companies' publications (unlike many of the reports we will examine later) often capture aggregate system behavior and large-scale statistical trends and indicate (often obliquely) that partitions are of concern in their deployments.

A team from the University of Toronto and Microsoft Research studied the behavior of network failures in several of Microsoft's datacenters.[12] They found an average failure rate of 5.2 devices per day and 40.8 links per day, with a median time to repair of approximately five minutes (and a maximum of one week). While the researchers note that correlating link failures and communication partitions is challenging, they estimate a median packet loss of 59,000 packets per failure. Perhaps more concerning is their finding that network redundancy improves median traffic by only 43%; that is, network redundancy does not eliminate common causes of network failure.

A joint study between researchers at UCSD and HP Labs examined the causes and severity of network failures in HP's managed networks by analyzing support ticket data (http://www.hpl.hp.com/techreports/2012/HPL-2012-101.pdf). Connectivity-related tickets accounted for 11.4% of support tickets (14% of which were of the highest priority level), with a median incident duration of 2 hours and 45 minutes for the highest-priority tickets and a median duration of 4 hours 18 minutes for all tickets.

Google's paper describing the design and operation of Chubby, their distributed lock manager, outlines the root causes of 61 outages over 700 days of operation across several clusters (http://research.google.com/archive/chubby-osdi06.pdf). Of the nine outages that lasted more than 30 seconds, four were caused by network maintenance and two were caused by "suspected network connectivity problems."

In "Design Lessons and Advice from Building Large Scale Distributed Systems" (http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf), Jeff Dean suggested a typical first year for a new Google cluster involves:

- Five racks go wonky (40–80 machines seeing 50% packet loss)
- Eight network maintenances (four might cause ~30-minute random connectivity losses)
- Three router failures (have to immediately pull traffic for an hour)

The degree to which networks are *actually* reliable in the real world is the subject of considerable and evolving debate.

While Google does not tell us much about the application-level consequences of their network partitions, Dean suggested they were of concern, citing the perennial challenge of creating "easy-to-use abstractions for resolving conflicting updates to multiple versions of a piece of state," useful for "reconciling replicated state in different data centers after repairing a network partition."

Amazon's Dynamo paper (http://bit.ly/1mDs0Yh) frequently cites the incidence of partitions as a key design consideration. Specifically, the authors note they rejected designs from "traditional replicated relational database systems" because they "are not capable of handling network partitions."

Yahoo! PNUTS/Sherpa was designed as a distributed database operating in geographically distinct datacenters. Originally, PNUTS supported a strongly consistent timeline consistency operation, with one master per data item. However, the developers noted that, in the event of network partitioning or server failures, this design decision was too restrictive for many applications:[16]

The first deployment of Sherpa supported the timeline-consistency model—namely, all replicas of a record apply all updates in the same order—and has API-level features to enable applications to cope with asynchronous replication. Strict adherence leads to difficult situations under network partitioning or server failures. These can be partially addressed with override procedures and local data replication, but in many circumstances, applications need a relaxed approach.

According to the same report, PNUTS now offers weaker consistency alternatives providing availability during partitions.

Datacenter Network Failures

Datacenter networks are subject to power failure, misconfiguration, firmware bugs, topology changes, cable damage, and malicious traffic. Their

failure modes are accordingly diverse.

As Microsoft's SIGCOMM paper suggests, redundancy does not always prevent link failure. When a power distribution unit failed and took down one of two redundant top-of-rack switches, Fog Creek lost service for a subset of customers on that rack but remained consistent and available for most users. However, the other switch in that rack *also* lost power for undetermined reasons. That failure isolated the two neighboring racks from each other, taking down all On Demand services.

During a planned network reconfiguration to improve reliability, Fog Creek suddenly lost access to its network.[10]

A network loop had formed between several switches. The gateways controlling access to the switch management network were isolated from each other, generating a split-brain scenario. Neither was accessible due to a...multi-switch BPDU (bridge protocol data unit) flood, indicating a spanning-tree flap. This is most likely what was changing the loop domain.

According to the BPDU standard, the flood should not have happened. But it did, and this deviation from the system's assumptions resulted in two hours of total service unavailability.

To address high latencies caused by a daisy-chained network topology, Github installed a set of aggregation switches in its datacenter (https://github.com/blog/1346-network-problems-last-friday). Despite a redundant network, the installation process resulted in bridge loops, and switches disabled links to prevent failure. This problem was quickly resolved, but later investigation revealed that many interfaces were still pegged at 100% capacity.

While that problem was under investigation, a misconfigured switch triggered aberrant automatic fault detection behavior: when one link was disabled, the fault detector disabled *all* links, leading to 18 minutes of downtime. The problem was traced to a firmware bug preventing switches from updating their MAC address caches correctly, forcing them to broadcast most packets to every interface.

In December 2012, a planned software update on an aggregation switch caused instability at Github (https://github.com/blog/1364-downtime-last-saturday). To collect diagnostic information, the network vendor killed a particular software agent running on one of the aggregation switches.

Github's aggregation switches are clustered in pairs using a feature called MLAG, which presents two physical switches as a single layer-2 device. The MLAG failure detection protocol relies on both Ethernet link state and a logical heartbeat message exchanged between nodes. When the switch agent was killed, it was unable to shut down the Ethernet link, preventing the still-healthy aggregation switch from handling link aggregation, spanning-tree, and other L2 protocols. This forced a spanning-tree leader election and reconvergence for all links, blocking all traffic between access switches for 90 seconds.

This 90-second network partition caused fileservers using Pacemaker and DRBD for HA failover to declare each other dead, and to issue STONITH (Shoot The Other Node In The Head) messages to one another. The network partition delayed delivery of those messages, causing some fileserver pairs to believe they were *both* active. When the network recovered, both nodes shot each other at the same time. With both nodes dead, files belonging to the pair were unavailable.

To prevent filesystem corruption, DRBD requires that administrators ensure the original primary node is still the primary node before resuming replication. For pairs where both nodes were primary, the ops team had to examine log files or bring each node online in isolation to determine its state. Recovering those downed fileserver pairs took five hours, during which Github service was significantly degraded.

Cloud Networks

Large-scale virtualized environments are notorious for transient latency, dropped packets, and full-blown network partitions, often affecting a particular software version or availability zone. Sometimes the failures occur between specific subsections of the provider's datacenter, revealing planes of cleavage in the underlying hardware topology.

In a comment on Call me maybe: MongoDB (http://aphyr.com/posts/284-call-me-maybe-mongodb), Scott Bessler observed exactly the same failure mode Kyle demonstrated in the Jepsen post:

[This scenario] happened to us today when EC2 West region had network issues that caused a network partition that separated PRIMARY from its 2 SECONDARIES in a 3 node replset. 2 hours later the old primary rejoined and rolled back everything on the new primary.

This partition caused two hours of write loss. From our conversations with large-scale MongoDB users, we gather that network events causing failover on EC2 are common. Simultaneous primaries accepting writes for multiple days are anecdotally common.

Outages can leave two nodes connected to the Internet but unable to see each other. This type of partition is especially dangerous, as writes to both sides of a partitioned cluster can cause inconsistency and lost data. Paul Mineiro reports exactly this scenario in a Mnesia cluster (http://bit.ly/1zrVxI1), which diverged overnight. The cluster's state was not critical, so the operations team simply nuked one side of the cluster. They concluded "the experience has convinced us that we need to prioritize up our network partition recovery strategy."

Network disruptions in EC2 can affect only certain groups of nodes. For instance, one report of a total partition between the front-end and back-end servers states that a site's servers lose their connections to all back-end instances for a few seconds, several times a month (https://forums.aws.amazon.com/thread.jspa?messageID=454155). Even though the disruptions were short, they resulted in 30–45 minute outages and a corrupted index for ElasticSearch. As problems escalated, the outages occurred "2 to 4 times a day."

On April 21, 2011, Amazon Web Services suffered unavailability for 12 hours,[2] causing hundreds of high-profile websites to go offline. As a part of normal AWS scaling activities, Amazon engineers had shifted traffic away from a router in the Elastic Block Store (EBS) network in a single U.S. East Availability Zone (AZ), but, due to incorrect routing policies:

...many EBS nodes in the affected Availability Zone were completely isolated from other EBS nodes in its cluster. Unlike a normal network interruption, this change disconnected both the pri-

mary and secondary network simultaneously, leaving the affected nodes completely isolated from one another.

The partition, coupled with aggressive failure-recovery code, caused a mirroring storm that caused network congestion and triggered a previously unknown race condition in EBS. EC2 was unavailable for approximately 12 hours, and EBS was unavailable or degraded for over 80 hours.

The EBS failure also caused an outage in Amazon's Relational Database Service. When one AZ fails, RDS is designed to fail over to a different AZ. However, 2.5% of multi-AZ databases in US-East failed to fail over due to a bug in the fail-over protocol.

This correlated failure caused widespread outages for clients relying on AWS. For example, Heroku reported between 16 and 60 hours of unavailability for their users' databases.

On July 18, 2013, Twilio's billing system, which stores account credits in Redis, failed.[19] A network partition isolated the Redis primary from all secondaries. Because Twilio did not promote a new secondary, writes to the primary remained consistent. However, when the primary became visible to the secondaries again, all secondaries simultaneously initiated a full resynchronization with the primary, overloading it and causing Redis-dependent services to fail.

The ops team restarted the Redis primary to address the high load. However, upon restart, the Redis primary reloaded an incorrect configuration file, which caused it to enter read-only mode. With all account balances at zero, and in read-only mode, every Twilio API call caused the billing system to automatically recharge customer credit cards. 1.1% of customers were overbilled over a period of 40 minutes. For example, Appointment Reminder reported that every SMS message and phone call they issued resulted in a $500 charge to their credit card, which stopped accepting charges after $3,500.

Twilio recovered the Redis state from an independent billing system—a relational datastore—and after some hiccups, restored proper service, including credits to affected users.

Hosting Providers

Running your own datacenter can be cheaper and more reliable than using public cloud infrastructure, but it means you have to be a network and server administrator. What about hosting providers, which rent dedicated or virtualized hardware to users and often take care of the network and hardware setup for you?

Freistil IT hosts their servers with a colocation/managed-hosting provider. Their monitoring system alerted Freistil to 50%–100% packet loss localized to a specific datacenter.[15] The network failure, caused by a router firmware bug, returned the next day. Elevated packet loss caused the GlusterFS distributed filesystem to enter split-brain undetected:

...we became aware of [problems] in the afternoon when a customer called our support hotline because their website failed to deliver certain image files. We found that this was caused by a split-brain situation...and the self-heal algorithm built into the Gluster filesystem was not able to resolve this inconsistency between the two data sets.

Repairing that inconsistency led to a "brief overload of the web nodes because of a short surge in network traffic."

Anecdotally, many major managed hosting providers experience network failures. One company running 100–200 nodes on a major hosting provider reported that in a 90-day period the provider's network went through five distinct periods of partitions. Some partitions disabled connectivity between the provider's cloud network and the public Internet, and others separated the cloud network from the provider's internal managed-hosting network.

A post to Linux-HA details a long-running partition between a Heartbeat pair (http://bit.ly/1k9Ym6V), in which two Linode VMs each declared the other dead and claimed a shared IP for themselves. Successive posts suggest further network problems: email messages failed to dispatch due to DNS resolution failure, and nodes reported the network unreachable. In this case, the impact appears to have been minimal, in part because the partitioned application was just a proxy.

Wide Area Networks

While we have largely focused on failures over local area networks (or near-local networks), wide area network (WAN) failures are also common, if less frequently documented. These failures are particularly interesting because there are often fewer redundant WAN routes and because systems guaranteeing high availability (and disaster recovery) often require distribution across multiple datacenters. Accordingly, graceful degradation under partitions or increased latency is especially important for geographically widespread services.

Researchers at the UCSD analyzed five years of operation in the CENIC wide-area network,[18] which contains over 200 routers across California. By cross-correlating link failures and additional external BGP and traceroute data, they discovered over 508 "isolating network partitions" that caused connectivity problems between hosts. Average partition duration ranged from six minutes for software-related failures to over 8.2 hours for hardware-related failures (median 2.7 and 32 minutes; 95th percentile of 19.9 minutes and 3.7 days, respectively).

PagerDuty designed their system to remain available in the face of node, datacenter, or even provider failure; their services are replicated between two EC2 regions and a datacenter hosted by Linode. On April 13, 2013, an AWS peering point in northern California degraded, causing connectivity issues for one of PagerDuty's EC2 nodes. As latencies between AWS availability zones rose, the notification dispatch system lost quorum and stopped dispatching messages entirely.

Even though PagerDuty's infrastructure was designed with partition tolerance in mind, correlated failures due to a shared peering point between two datacenters caused 18 minutes of unavailability, dropping inbound API requests and delaying queued pages until quorum was re-established.

Global Routing Failures

Despite the high level of redundancy in Internet systems, some network failures take place on a global scale.

CloudFlare runs 23 datacenters with redundant network paths and anycast failover. In response to a DDoS attack against one of their customers, the CloudFlare operations team deployed a new firewall rule to drop packets of a specific size.[17] Juniper's FlowSpec protocol propagated that rule to all CloudFlare edge routers, but then:

What should have happened is that no packet should have matched that rule because no packet was actually that large. What happened instead is that the routers encountered the rule and then proceeded to consume all their RAM until they crashed.

Recovering from the failure was complicated by routers that failed to reboot automatically and by inaccessible management ports.

Even though some data centers came back online initially, they fell back over again because all the traffic across our entire network hit them and overloaded their resources.

CloudFlare monitors its network carefully, and the operations team had immediate visibility into the failure. However, coordinating globally distributed systems is complex, and calling on-site engineers to find and reboot routers by hand takes time. Recovery began after 30 minutes, and was complete after an hour of unavailability.

A firmware bug introduced as a part of an upgrade in Juniper Networks's routers caused outages in Level 3 communications' networking backbone in 2011. This subsequently knocked services including Time Warner Cable, RIM BlackBerry, and several U.K. Internet service providers offline.

There have been several global Internet outages related to BGP misconfiguration. Notably, in 2008, Pakistan Telecom, responding to a government edict to block YouTube.com, incorrectly advertised its (blocked) route to other providers, which hijacked traffic from the site and briefly rendered it unreachable.

In 2010, a group of Duke University researchers achieved a similar effect by testing an experimental flag in the BGP protocol (http://bit.ly/1rbAl4j). Similar incidents occurred in 2006, knocking sites such as Martha Stewart Living and the *New York Times* offline; in 2005, where a misconfiguration in Turkey attempted in a redirect for the entire Internet; and in 1997.

Despite the high level of redundancy in Internet systems, some network failures take place on a global scale.

NICs and Drivers

Unreliable networking hardware and/or drivers are implicated in a broad array of partitions.

As a classic example of NIC unreliability, Marc Donges and Michael Chan describe how their popular Broadcom BCM5709 chip dropped inbound but not outbound packets (http://www.spinics.net/lists/netdev/msg210485.html). The primary server was unable to service requests, but, because it could still send heartbeats to its hot spare, the spare considered the primary alive and refused to take over. Their service was unavailable for five hours and did not recover without a reboot.

Sven Ulland followed up, reporting the same symptoms with the BCM5709S chipset on Linux 2.6.32-41squeeze2. Despite pulling commits from mainline that supposedly fixed a similar set of issues with the bnx2 driver, Ulland's team was unable to resolve the issue until version 2.6.38.

As a large number of servers shipped the BCM5709, the larger impact of these firmware bugs was widely observed. For instance, the 5709 had a bug in their 802.3x flow control, leading to extraneous PAUSE frames when the chipset crashed or its buffer filled up. This problem was magnified by the BCM56314 and BCM56820 switch-on-a-chip devices (found in many top-of-rack switches), which, by default, sent PAUSE frames to any interface communicating with the offending 5709 NIC. This led to cascading failures on entire switches or networks.

The bnx2 driver could also cause transient or flapping network failures, as described in an ElasticSearch failure report. Meanwhile, the Broadcom 57711 was notorious for causing high latencies under load with jumbo frames, a particularly thorny issue for ESX users with iSCSI-backed storage.

A motherboard manufacturer failed to flash the EEPROM correctly for its Intel 82574–based system. The result was a very-hard-to-diagnose error in which an inbound SIP packet of a particular structure would disable the NIC.[14] Only a cold restart would bring the system back to normal.

After a scheduled upgrade, CityCloud noticed unexpected network failures in two distinct GlusterFS pairs, followed by a third.[6] Suspecting link aggregation, CityCloud disabled the feature on its switches and allowed self-healing operations to proceed.

Roughly 12 hours later, the network failures returned. CityCloud identified the cause as a driver issue and updated

the downed node, returning service. However, the outage resulted in data inconsistency between GlusterFS pairs and data corruption between virtual machine file systems.

Application-Level Failures

Not all asynchrony originates in the physical network. Sometimes dropped or delayed messages are a consequence of crashes, program errors, OS scheduler latency, or overloaded processes. The following studies highlight the fact that communication failures—wherein the system delays or drops messages—can occur at any layer of the software stack, and designs that expect synchronous communication may behave unexpectedly during periods of asynchrony.

Bonsai.io (http://www.bonsai.io/blog/2013/03/05/outage-post-mortem) discovered high CPU and memory use on an ElasticSearch node combined with difficulty connecting to various cluster components, likely a consequence of an "excessively high number of expensive requests being allowed through to the cluster."

Upon restarting the servers, the cluster split into two independent components. A subsequent restart resolved the split-brain behavior, but customers complained they were unable to delete or create indices. The logs revealed that servers were repeatedly trying to recover unassigned indices, which "poisoned the cluster's attempt to service normal traffic which changes the cluster state." The failure led to 20 minutes of unavailability and six hours of degraded service.

Stop-the-world garbage collection and blocking for disk I/O can cause runtime latencies on the order of seconds to minutes. As Searchbox IO and several other production users have found, GC pressure in an ElasticSearch cluster can cause secondary nodes to declare a primary dead and to attempt a new election (https://github.com/elasticsearch/elasticsearch/issues/2488). Due to non-majority quorum configuration, ElasticSearch elected two different primaries, leading to inconsistency and downtime. Surprisingly, even with majority quorums, due to protocol design, ElasticSearch does not currently prevent simultaneous master election; GC pauses and high IO_WAIT times due to I/O can cause split-brain behavior, write loss, and index corruption.

Stop-the-world garbage collection and blocking for disk I/O can cause runtime latencies on the order of seconds to minutes.

In 2012, a routine database migration caused unexpectedly high load on the MySQL primary at Github.[13] The cluster coordinator, unable to perform health checks against the busy MySQL server, decided the primary was down and promoted a secondary. The secondary had a cold cache and performed poorly, causing failover back to the original primary. The operations team manually halted this automatic failover and the site appeared to recover.

The next morning, the operations team discovered the standby MySQL node was no longer replicating changes from the primary. Operations decided to disable the coordinator's maintenance mode and allow the replication manager to fix the problem. Unfortunately, this triggered a segfault in the coordinator, and, due to a conflict between manual configuration and the automated replication tools, github.com was rendered unavailable.

The partition caused inconsistency in the MySQL database—both between the secondary and primary, and between MySQL and other data stores such as Redis. Because foreign key relationships were not consistent, Github showed private repositories to the wrong users' dashboards and incorrectly routed some newly created repositories.

When a two-node cluster partitions, there are no cases in which a node can reliably declare itself to be the primary. When this happens to a DRBD filesystem, as one user reported (http://bit.ly/1nbv4E), both nodes can remain online and accept writes, leading to divergent filesystem-level changes.

Short-lived failures can lead to long outages. In a Usenet post to novell.support.cluster-services, an admin reports that a two-node failover cluster running Novell NetWare experienced transient network outages. The secondary node eventually killed itself, and the primary (though still running) was no longer reachable by other hosts on the network. The post goes on to detail a series of network partition events correlated with backup jobs!

One VoltDB user reports regular network failures causing replica divergence (http://bit.ly/1mDeC4d) but also indicates their network logs included

no dropped packets. Because this cluster had not enabled split-brain detection, both nodes ran as isolated primaries, causing significant data loss.

Sometimes, nobody knows why a system partitions. This RabbitMQ failure seems like one of those cases: few retransmits, no large gaps between messages, and no clear loss of connectivity between nodes (http://bit.ly/1qZROze). Increasing the partition detection timeout to two minutes reduced the frequency of partitions but did not prevent them altogether.

Another EC2 split-brain (http://bit.ly/1mDeIZA): a two-node cluster failed to converge on "roughly 1 out of 10 startups" when discovery messages took longer than three seconds to exchange. As a result, both nodes would start as primaries with the same cluster name. Since ElasticSearch does not demote primaries automatically, split-brain persisted until administrators intervened. Increasing the discovery timeout to 15 seconds resolved the issue.

There are a few scattered reports of Windows Azure partitions, such as this account of a RabbitMQ cluster that entered split-brain on a weekly basis (http://bit.ly/1sCN4Nw). There's also this report of an ElasticSearch split-brain (http://bit.ly/U5xAFS), but since Azure is a relative newcomer compared to EC2, descriptions of its network reliability are limited.

Where Do We Go From Here?

This article is meant as a reference point to illustrate that, according to a wide range of (often informal) accounts, communication failures occur in many real-world environments. Processes, servers, NICs, switches, and local and wide area networks can all fail, with real economic consequences. Network outages can suddenly occur in systems that have been stable for months at a time, during routine upgrades, or as a result of emergency maintenance. The consequences of these outages range from increased latency and temporary unavailability to inconsistency, corruption, and data loss. Split-brain is not an academic concern: it happens to all kinds of systems, sometimes for days on end. Partitions deserve serious consideration.

On the other hand, some networks really are reliable. Engineers at major financial firms have anecdotally reported that despite putting serious effort into designing systems that gracefully tolerate partitions, their networks rarely, if ever, exhibit partition behavior. Cautious engineering and aggressive network advances (along with lots of money) can prevent outages. Moreover, in this article, we have presented failure scenarios; we acknowledge it is much more difficult to demonstrate that network failures have not occurred!

However, not all organizations can afford the cost or operational complexity of highly reliable networks. From Google and Amazon (who operate commodity and/or low-cost hardware due to sheer scale) to one-person startups built on shoestring budgets, communication-isolating network failures are a real risk, in addition to the variety of other failure modes (including human error) that real-world distributed systems face.

It is important to consider this risk before a partition occurs, because it is much easier to make decisions about partition behavior on a whiteboard than to redesign, reengineer, and upgrade a complex system in a production environment—especially when it is throwing errors at your users. For some applications, failure *is* an option, but you should characterize and explicitly account for it as a part of your design. And finally, given the additional latency[1] and coordination benefits[4] of partition-aware designs, you might just find that accounting for these partitions delivers benefits in the average case as well.

We invite you to contribute your own experiences with or without network partitions. Open a pull request on https://github.com/aphyr/partitions-post (which, incidentally, contains all references), leave a comment, write a blog post, or release a post-mortem. Data will inform this conversation, future designs, and, ultimately, the availability of the systems we all depend on. C

Q Related articles on queue.acm.org

Eventual Consistency Today: Limitations, Extensions, and Beyond
Peter Bailis and Ali Ghodsi
http://queue.acm.org/detail.cfm?id=2462076

The Antifragile Organization
Ariel Tseitlin
http://queue.acm.org/detail.cfm?id=2499552

Self-Healing Networks
Robert Poor, Cliff Bowman and Charlotte Burgess Auburn
http://queue.acm.org/detail.cfm?id=864027

References

1. Abadi, D. Consistency trade-offs in modern distributed database system design: CAP is only part of the story. *Computer 45* (2 (2012), 37–42; http://dl.acm.org/citation.cfm?id=2360959.
2. Amazon Web Services. Summary of the Amazon EC2 and Amazon RDS service disruption in the US East region, 2011; http://aws.amazon.com/message/65648/.
3. Bailis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J.M. and Stoica, I. Highly available transactions: virtues and limitations. In *Proceedings of VLDB 2014* (to appear); http://www.bailis.org/papers/hat-vldb2014.pdf.
4. Bailis, P., Fekete, A., Franklin, M.J., Ghodsi, A., Hellerstein, J.M. and Stoica, I. Coordination-avoiding database systems, 2014; http://arxiv.org/abs/1402.2237
5. Bailis, P. and Ghodsi, A. Eventual consistency today: Limitations, extensions, and beyond. *ACM Queue 11*, 3 (2013); http://queue.acm.org/detail.cfm?id=2462076 .
6. CityCloud, 2011; https://www.citycloud.eu/cloud-computing/post-mortem/.
7. Davidson, S.B., Garcia-Molina, H. and Skeen, D. Consistency in a partitioned network: A survey. *ACM Computing Surveys 17*, 3 (1985), 341–370; http://dl.acm.org/citation.cfm?id=5508.
8. Dwork, C., Lynch, M. and Stockmeyer, L. Consensus in the presence of partial synchrony. *JACM 35*, 2 (1988); 288–323. http://dl.acm.org/citation.cfm?id=42283.
9. Fischer, M.J., Lynch, N.A., Patterson, M.S. Impossibility of distributed consensus with one faulty process. *JACM 32*, 2 (1985), 374–382; http://dl.acm.org/citation.cfm?id=214121
10. Fog Creek Software. May 5–6 network maintenance post-mortem; http://status.fogcreek.com/2012/05/may-5-6-network-maintenance-post-mortem.html.
11. Gilbert, S. and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News 33*, 2 (2002), 51–59; http://dl.acm.org/citation.cfm?id=564601.
12. Gill, P., Jain, N., Nagappan, N. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proceedings of SIGCOMM '11*; http://research.microsoft.com/enus/um/people/navendu/papers/sigcomm11netwiser.pdf.
13. Github. Github availability this week, 2012; https://github.com/blog/1261-github-availability-this-week.
14. Kielhofner, K. Packets of death; http://blog.krisk.org/2013/02/packets-of-death.html.
15. Lillich, J. Post mortem: Network issues last week; http://www.freistil.it/2013/02/post-mortem-network-issues-last-week/.
16. Narayan, P.P.S. Sherpa update, 2010; https://developer.yahoo.com/blogs/ydn/sherpa-7992.html#4.
17. Prince, M. Today's outage post mortem, 2013; http://blog.cloudflare.com/todays-outage-post-mortem-82515.
18. Turner, D., Levchenko, K., Snoeren, A. and Savage, S. California fault lines: Understanding the causes and impact of network failures. In *Proceedings of SIGCOMM '10*; http://cseweb.ucsd.edu/~snoeren/papers/cenic-sigcomm10.pdf.
19. Twilio. Billing incident post-mortem: breakdown, analysis and root cause; http://www.twilio.com/blog/2013/07/billing-incident-post-mortem.html.

Peter Bailis is a graduate student of computer science and a member of the AMPLab and BOOM projects at UC Berkeley. He studies database and distributed systems and blogs at http://bailis.org/blog and tweets as @pbailis.

Kyle Kingsbury (@aphyr) is the author of Riemann, Timelike, and a number of other open source packages. He also verifies distributed systems' safety claims as part of the Jepsen project.