The network is reliable

2013/06/02
Jepsen Networking

I've been discussing Jepsen and partition tolerance with Peter Bailis over the past few weeks, and I'm honored to present this post as a collaboration between the two of us. We'd also like to extend our sincere appreciation to everyone who contributed their research and experience to this piece.

Network partitions are a contentious subject. Some claim that modern networks are reliable and that we are too concerned with designing for *theoretical* failure modes. They often accept that single-node failures are common but argue that we can reliably detect and handle them. Conversely, others subscribe to Peter Deutsch's Fallacies of Distributed Computing and disagree. They attest that partitions do occur in their systems, and that, as James Hamilton of Amazon Web Services neatly summarizes, "network partitions should be rare but net gear continues to cause more issues than it should." The answer to this debate radically affects the design of distributed databases, queues, and applications. So who's right?

A key challenge in this dispute 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 an organization has 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 they are understood at all.

As a result, much of what we know about the failure modes of real-wold distributed systems is founded on guesswork and rumor. Sysadmins and developers will swap stories over beers, but detailed, public postmortems and comprehensive surveys of network availability are few and far between. In this post, we'd like to bring a few of these stories together. We believe this is a first step towards a more open and honest discussion of real-world partition behavior, and, ultimately, 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 nodes. Of all of the data we have collected, these reports best summarize operation in the large, distilling the experience of operating what are likely the biggest distributed systems ever deployed. Their publications (unlike many of the case studies we will examine later) often capture aggregate system behavior and large-scale statistical trends, and indicate (often obliquely) that partitions are a significant concern in their deployments.

The Microsoft Datacenter Study

A team from the University of Toronto and Microsoft Research studied the behavior of network failures in several of Microsoft's datacenters. 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 up to 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 many common causes of network failure.

HP Enterprise Managed Networks

A joint study between researchers at University of California, San Diego and HP Labs examined the causes and severity of network failures in HP's managed networks by analyzing support ticket data. "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 and a median duration of 4 hours 18 minutes for all priorities.

Google Chubby

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. Of the nine outages that lasted greater than 30 seconds, four were caused by network maintenance and two were caused by "suspected network connectivity problems."

Google's Design Lessons from Distributed Systems

In Design Lessons and Advice from Building Large Scale Distributed Systems, Jeff Dean suggests that a typical first year for a new Google cluster involves:

- 5 racks going wonky (40-80 machines seeing 50% packet loss)
- 8 network maintenances (4 might cause ~30-minute random connectivity losses)
- 3 router failures (have to immediately pull traffic for an hour)

While Google doesn't tell us much about the application-level consequences of their network partitions, "Lessons From Distributed Systems" suggests they are a significant concern, citing the challenge of "[e]asy-to-use abstractions for resolving conflicting updates to multiple versions of a piece of state" as useful in "reconciling replicated state in different data centers after repairing a network partition."

Amazon Dynamo

Amazon's Dynamo paper frequently cites the incidence of partitions as a driving design consideration. Specifically, the authors note that they rejected designs from "traditional replicated relational database systems" because they "are not capable of handling network partitions."

Yahoo! PNUTS/Sherpa

Yahoo! PNUTS/Sherpa was designed as a distributed database operating out of multiple, geographically distinct sites. 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:

> 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. "

➤ Application-level failures

Not all partitions originate in the physical network. Sometimes dropped or delayed messages are a consequence of crashes, race conditions, OS scheduler latency, or overloaded processes. The following studies highlight the fact that partitions–wherein the system delays or drops messages–can occur at any layer of the software stack.

CPU use and service contention

Bonsai.io 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."

They restarted the cluster, but on restarting the cluster partitioned itself into two independent components. A subsequent cluster restart resolved the partition, 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.

Bonsai concludes by noting that large-scale ElasticSearch clusters should use dedicated nodes which handle routing and leader election without serving normal requests for data, to prevent partitions under heavy load. They also emphasize the importance of request throttling and setting proper quorum values.

Long GC pauses and IO

Stop-the-world garbage collection and blocking for disk IO can cause runtime latencies on the order of seconds to minutes. As Searchbox IO and dozens of 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. Because their configuration used a low value of `zen.minimum_master_nodes`, ElasticSearch was able to elect two simultaneous primaries, leading to inconsistency and downtime. Even with `minimum_master_nodes` larger than a majority, ElasticSearch does not prevent nodes from taking part in multiple network components; GC pauses and high IO_WAIT times due to IO can cause split brain, write loss, and index corruption.

MySQL overload and a Pacemaker segfault

Github relies heavily on Pacemaker and Heartbeat: programs which coordinate cluster resources between nodes. They use Percona Replication Manager, a resource agent for Pacemaker, to replicate their MySQL database between three nodes.

On September 10th, 2012, a routine database migration caused unexpectedly high load on the MySQL primary. Percona Replication Manager, unable to perform health checks against the busy MySQL instance, decided the primary was down and promoted a secondary. The secondary had a cold cache and performed poorly. Normal query load on the node caused it to slow down, and Percona failed *back* to the original primary. The operations team put Pacemaker into maintenance-mode, temporarily halting automatic failover. The site appeared to recover.

The next morning, the operations team discovered that the standby MySQL node was no longer replicating changes from the primary. Operations decided to disable Pacemaker's maintenance mode to allow the replication manager to fix the problem.

> Upon attempting to disable maintenance-mode, a Pacemaker segfault occurred that resulted in a cluster state partition. After this update, two nodes (I'll call them 'a' and 'b') rejected most messages from the third node ('c'), while the third node rejected most messages from the other two. Despite having configured the cluster to require a majority of machines to agree on the state of the cluster before taking action, two simultaneous master election decisions were attempted without proper coordination. In the first cluster, master election was interrupted by messages from the second cluster and MySQL was stopped.
>
> In the second, single-node cluster, node 'c' was elected at 8:19 AM, and any subsequent messages from the other two-node cluster were discarded. As luck would have it, the 'c' node was the node that our operations team previously determined to be out of date. We detected this fact and powered off this out-of-date node at 8:26 AM to end the partition and prevent further data drift, taking down all production database access and thus all access to github.com.

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

Github thought carefully about their infrastructure design, and were still surprised by a complex interaction of partial failures and software bugs. As they note in the postmortem:

> … if any member of our operations team had been asked if the failover should have been performed, the answer would have been a resounding **no**.

Distributed systems are *hard*.

➤ NICs and drivers

BCM5709 and friends

Unreliable NIC hardware or drivers are implicated in a broad array of partitions. Marc Donges and Michael Chan bring us a thrilling report of the popular Broadcom BCM5709 chipset abruptly dropping inbound *but not outbound* packets to a machine. Because the NIC dropped inbound packets, the node was unable to service requests. However, because it could still *send* heartbeats to its hot spare via keepalived, the spare considered the primary alive and refused to take over. The 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 which supposedly fixed a similar set of issues with the bnx2 driver, they were unable to resolve the issue until version 2.6.38.

Since Dell shipped a large number of servers with the BCM5709, the impact of these firmware bugs was widely observed. For instance, the 5709 and some chips had a bug in their 802.3x flow control code causing them to spew 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 (a component in a number of Dell's top-of-rack switches), which, by default, spewed PAUSE frames at *every* interface trying to communicate 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 this ElasticSearch split brain report. Meanwhile, the the Broadcom 57711 was notorious for causing extremely high latencies under load with jumbo frames, a particularly thorny issue for ESX users with iSCSI-backed storage.

Intel 82574 Packet of Death

A motherboard manufacturer failed to flash the EEPROM correctly for their Intel 82574 based system. The result was a very hard to diagnose error where an inbound SIP packet of a particular structure would disable the NIC. Only a cold restart would bring the system back to normal.

A GlusterFS partition caused by a driver bug

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

Roughly 12 hours later, the network failures returned on one node. 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:

> As the servers lost storage abruptly there were certain types of Gluster issues where files did not match each other on the two nodes in each storage pair. There were also some cases of data corruption in the VMs filesystems due to VMs going down in an uncontrolled way.

➤ Datacenter network failures

Individual network interfaces can fail, but they typically appear as single-node outages. Failures located in the physical network are often more nefarious. Switches are subject to power failure, misconfiguration, firmware bugs, topology changes, cable damage, and malicious traffic. Their failure modes are accordingly diverse:

Power failure on both redundant switches

As Microsoft's SIGCOMM paper suggests, redundancy doesn't 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 one another, taking down all On Demand services.

Switch split-brain caused by BPDU flood

During a planned network reconfiguration to improve reliability, Fog Creek suddenly lost access to their network.

> 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 were accessible due to a sudden traffic flood.
>
> The flood was the result of 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 *shouldn't have happened*. But it did, and this deviation from the system's assumptions resulted in two hours of total service unavailability.

Bridge loops, misconfiguration, broken MAC caches

In an effort to address high latencies caused by a daisy-chained network topology, Github installed a set of aggregation

switches in their datacenter. 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 investigating that problem, a misconfigured switch triggered aberrant automatic fault detection behavior: when one link was disabled, the fault detector disabled *all* links. This caused 18 minutes of hard downtime. The problem was later traced to a firmware bug preventing switches from updating their MAC address caches correctly, which forced them to broadcast most packets to every interface.

Mystery RabbitMQ partitions

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. Upping the partition detection timeout to 2 minutes reduced the frequency of partitions but didn't prevent them altogether.

DRBD split-brain

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, both nodes can remain online and accept writes, leading to divergent filesystem-level changes. The only realistic option for resolving these kinds of conflicts is to discard all writes not made to a selected component of the cluster.

A NetWare split-brain

Short-lived failures can lead to long outages. In this Usenet post to novell.support.cluster-services, an admin reports their 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!

MLAG, Spanning Tree, and STONITH

Github writes great postmortems, and this one is no exception. On December 22nd, 2012, a planned software update on an aggregation switch caused some mild instability during the maintenance window. In order to collect diagnostic information about the instability, 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. Unlucky timing confused the MLAG takeover, preventing the still-healthy agg switch from handling link aggregation, spanning-tree, and other L2 protocols as normal. This forced a spanning-tree leader election and reconvergence for all links, *blocking all traffic between access switches for 90 seconds*.

The 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 the node online in isolation to determine its state. Recovering those downed fileserver pairs took five hours, during which Github service was significantly degraded.

➤ Hosting providers

Running your own datacenter can be cheaper and more reliable than using public cloud infrastructure, but it also 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?

An undetected GlusterFS split-brain

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. 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:

> Unfortunately, the malfunctioning network had caused additional problems which we became aware of 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 on the storage cluster "stor02" where changes made on node "stor02b" weren't reflected on "stor02a" 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."

An anonymous hosting provider

From what we can gather informally, *all* the major managed hosting providers experience regular 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. The failures caused unavailability, but because this company wasn't running any significant distributed systems *across* those partitioned networks, there was no observed inconsistency or data loss.

Pacemaker/Heartbeat split-brain

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

➤ 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.

An isolated MongoDB primary on EC2

In a comment on Call me maybe: MongoDB, Scott Bessler observed exactly the same failure mode Kyle demonstrated in the Jepsen post:

> "Prescient. The w=safe scenario you show (including extra fails during rollback/re-election) 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. Our bad for not using w=majority."

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 not unknown.

Mnesia split-brain on EC2

EC2 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. That's exactly what happened to this Mnesia cluster, which diverged overnight. Their state wasn't critical, so the operations team simply nuked one side of the cluster. They conclude: "the experience has convinced us that we need to prioritize up our network partition recovery strategy".

EC2 instability causing MongoDB and ElasticSearch unavailability

Network disruptions in EC2 can affect only certain groups of nodes. For instance, this report of a total partition between the frontend and backend stacks states that their the web servers lose their connections to all backend instances for a few seconds, several times a month. Even though the disruptions were short, cluster convergence resulted in 30-45 minute outages and a corrupted index for ElasticSearch. As problems escalated, the outages occurred "2 to 4 times a day."

VoltDB split-brain on EC2

One VoltDB user reports regular network failures causing replica divergence but also indicates that 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.

ElasticSearch discovery failure on EC2

Another EC2 split-brain: 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 doesn't demote primaries automatically, split-brain persisted until administrators intervened. Upping the discovery timeout to 15 seconds resolved the issue.

RabbitMQ and ElasticSearch on Windows Azure

There are a few scattered reports of Windows Azure partitions, such as this account of a RabbitMQ cluster which entered split-brain on a weekly basis. There's also this report of an ElasticSearch split-brain, but since Azure is a relative newcomer compared to EC2, descriptions of its network reliability are limited.

AWS EBS outage

On April 21st, 2011, Amazon Web Services went down for over 12 hours, causing hundreds of high-profile web sites to go offline. As a part of normal AWS scaling activities, Amazon engineers shifted traffic away from a router in the Elastic Block Store (EBS) network in a single US-East Availability Zone (AZ).

> The traffic shift was executed incorrectly and rather than routing the traffic to the other router on the primary network, the traffic was routed onto the lower capacity redundant EBS network. For a portion of the EBS cluster in the affected Availability Zone, this meant that they did not have a functioning primary or secondary network because traffic was purposely shifted away from the primary network and the secondary network couldn't handle the traffic level it was receiving. As a result, 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 primary 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, which led to network congestion and triggered a previously unknown race condition in EBS. EC2 was unavailable for roughly 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 "stuck" IO.

> The primary cause was that the rapid succession of network interruption (which partitioned the primary from the secondary) and "stuck" I/O on the primary replica triggered a previously un-encountered bug. This bug left the primary replica in an isolated state where it was not safe for our monitoring agent to automatically fail over to the secondary replica without risking data loss, and manual intervention was required. "

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.

Isolated Redis primary on EC2

On July 18, 2013, Twilio's billing system, which stores account credits in Redis, failed.. A network partition isolated the Redis primary from all billing 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 initiated a full resynchronization with the primary simultaenously. This overloaded the primary, causing services which relied on the Redis primary to fail.

The ops team restarted the Redis primary to address the high load; but on restarting, the Redis primary reloaded an incorrect configuration file which caused it to become a slave of itself. The primary entered read-only mode, which stopped all billing system writes. With all account balances at zero, and read-only, every Twilio call caused the billing system to automatically re-charge customer credit cards. 1.1% of customers were overbilled, for roughly 40 minutes. Appointment Reminder, for example, reported that every SMS message and phone call they issued resulted in a $500 charge to their credit card, which stopped accepting charges after $3500.

Twilio recovered the billing state from an independent billing system–a relational datastore–and after some hiccups,

restored proper service, including credits to affected users.

➤ WAN links

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.

PagerDuty

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.

CENIC Study

Researchers at the University of California, San Diego quantitatively analyzed five years of operation in the CENIC wide-area network, which contains over two hundred 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 6 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).

➤ Global routing failures

Despite the high level of redundancy in internet systems, some network failures take place on a globally distributed scale.

Cloudflare

CloudFlare runs 23 datacenters with redundant network paths and anycast failover. In response to a DDoS attack against one of their customers, their operations team deployed a new firewall rule to drop packets of a specific size. 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 which failed to reboot automatically, and 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 their network carefully and the ops 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.

Juniper routing bug

A firmware bug introduced as a part of an upgrade in Juniper Networks' routers caused outages in Level 3 Communications' networking backbone. This subsequently knocked services like Time Warner Cable, RIM BlackBerry, and several UK internet service providers offline.

Global BGP outages

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 provides, 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. Similar incidents occurred in 2006 (knocking sites like 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.

Where do we go from here?

This post is meant as a reference point–to illustrate that, according to a wide range of accounts, partitions occur in many real-world environments. Processes, servers, NICs, switches, local and wide area networks can all fail, and the resulting economic consequences are real. Network outages can suddenly arise in systems that are 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 report that despite putting serious effort into designing systems that gracefully tolerate partitions, their networks rarely, if ever, exhibit partition behavior. Cautious engineering (and lots of money) can prevent outages.

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-man startups built on shoestring budgets, communication-isolating network failures are a real risk.

It's important to consider this risk *before* a partition occurs–because it's much easier to make decisions about partition tolerance on a whiteboard than to redesign, re-engineer, and upgrade a complex system in a production environment–especially when it's 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.

We invite you to contribute your own experiences with or without network partitions. Open a pull request on https://github.com/aphyr/partitions-post, 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.

chiradeep, on 2013/06/05

There's also the AWS S3 outage from 2008 due to bit corruption in the gossip messages [http://status.aws.amazon.com/s3-20080720.html]. As a result, servers were not able to determine the true system state.

Marcel Kincaid, on 2013/06/05

Nice survey article.

ChadF, on 2013/06/05

Very informative (definitely makes you think if pondering data/service recovery methods).

And to paraphrase a wise man.. "With great STONITH power comes great responsibility".

chiradeep, on 2013/06/06

Also the usually-reliable long distance telephone network was down in 1990 http://www.phworld.org/history/attcrash.htm

Sean, on 2013/06/10

Regarding the Duke University "experiment": it was not the experiment, which was fully

within the BGP spec but the bug in a specific Cisco systems that caused the problem. http://www.cisco.com/en/US/products/csa/cisco-sa-20100827-bgp.html

Further details can be found at: https://labs.ripe.net/Members/erik/ripe-ncc-and-duke-university-bgp-experiment

Alan Robertson, on 2013/06/25

A network partition doesn't even have to be caused by network issues. Red Hat 2.6.18-2.6.20 had scheduler bugs that would stop heartbeats from being sent for hours - and it only happened under very light load. I've seen similar issues more recently in RHEL6 with some Linux boot parameters related to whether to sleep or go into the idle loop when there was nothing to do.

One feature we put into Linux-HA which got pulled after I left the project was the idea of having a single node be allowed to act as a tie-breaker for dozens or even hundreds of clusters. The original reason for its creation was split-site clusters, but it would have certainly solved many of the simultaneous STONITH shootings that Github saw.

Brian Merritt, on 2013/07/24

Another partition in the wild: http://www.twilio.com/blog/2013/07/billing-incident-post-mortem.html

Ben McCann, on 2013/10/01

Would love to see a Jepsen post for ElasticSearch!

Post a Comment

Name

Email

Http

Supports github-flavored markdown for [links](http://foo.com/), *emphasis*, _underline_, `code`, and > blockquotes. Use ```clj on its own line to start a Clojure code block, and ``` to end the block.

Post Comment