Data Center Networks Overview

Jiachang Chen¹, Hai Xi², Yuke Gao³, Yilun Liang⁴

¹Shandong Experimental High School, Jinan, Shandong 250001, China;
²Shenzhen Houde Academy, Shenzhen, Guangdong 518000, China;
³College of Science, China Agricultural University, Beijing 100083, China;
⁴Christ School, Arden, NC, 28704, United States.

Abstract

This paper talks about systems and protocols related to the data center along with problems occurring with running data centers and solutions addressing those errors. To begin with, basic concepts of the data center is introduced by presenting the design and evaluation of a representative system——Portland——which is a specifically designed protocol for data centers with efficient routing, loop-avoided forwarding, and effective detection and recovery about faults. After introducing some basic ideas and topology about the data center, problems regarding data center are discussed; specifically, the Google and Facebook Data Center is analyzed. Google Data Center's four-clusterpost network could no longer satisfy the demand for the bandwidth, a necessary solution to scale out, and a cost-effective means to update. On the other hand, the operational data of production network infrastructure at Facebook is presented to show the roles that the reliability plays in large scale data centers’ network. Lastly, after exquisitely describing the nature, pros and cons, and the characteristics of Data Centers, an application developed upon the data centers is interpreted: CrystalNet, a network emulator, is discussed, which can generate a virtue network functioning like a real network, allowing network software operators to test or validate their ideas and software tools on the virtue network before implementing them to the real network.

Keywords

Portland; CrystalNet; Data Center; Reliability.

1. Introduction

Nowadays, data centers are embraced by more and more companies and scholars as a solution to access efficient computing, storage, and other functions, leading the arrival of "mega data centers" containing applications possessing on a huge amount of hosts [1]. As the significance of data centers attains more weight, the data center networks, which are like the soul in data centers, are increasingly important. To comprehend data center networks better, several aspects of data center networks are introduced.

1.1 Functions

A data center is a conglomeration of resources, such as computational, storage, and network resources, and is interconnected by data center network (DCN) which plays an essential role in a data center. The job of the data center network is to establish and interconnect all physical and network-based machines into a data center facility. It can build a digital web that connects data center infrastructure nodes and enables them to transfer data both internally and externally. DCN needs to be capable of
connecting hundreds or more physical machines and handling the increasing demands for cloud computing.

1.2 Topology

Current data centers are usually comprised of copious physical machines arranged in several structures to effectively handle a considerable amount of data transportation and other missions. The data center networks are typically deployed in a topology similar to a fat tree, which is deeply hierarchical. Figure 1 demonstrates what is a fat tree topology. The 4-port switches are classified into three distinct layers: edge switches, aggregation switches, and core switches. In general, a three-stage fat tree topology, which is based on k-port switches, can hold non-blocking communication among \( \frac{k^3}{4} \) end hosts by \( \frac{5k^2}{4} \) individual k-port switches. The whole topology is divided into k individual pods; each pod supports the non-blocking operation among \( \frac{k^2}{4} \) hosts [2]. In this paper, the design of PortLand and the test for it are based on the fat tree topology.

1.3 Forwarding

Data center networks currently can utilize several forwarding protocols, such as OSPF [3]. To meet increasing demand for virtual machines’ migrations, the data centers use a layer 2 network based on flat MAC addresses to communicate.

1.4 Requirements

To obtain a highly efficient data center network, these requirements are vital.

a) Easy migrations of virtual machines: There is an increase in the number of end-host virtualization in data centers, adding additional requirements such as preventing virtual machines migrating at layer 3, because, otherwise, it will interrupt the existing TCP connections.

b) The ability to communicate: Any end host in a data center should be capable of communicating with others both in and out of the data center with fewer delays.

c) High reliability: To keep web services running continuously, the network infrastructure must possess high reliability. Otherwise, there will be troubles costing huge efforts to recovery. In order to build and operate highly-available web services, engineers and operators need to have a decent knowledge of network reliability, giving web services the ability to tolerate faults and also recover them.

d) Testbed at a large scale: Hitherto, as the data center is likely to contain more and more switches and physical machines, a large-scale and efficient software (discussed in section 5) to test the performance of a data center is increasingly important to prevent from losing a huge amount of customers.

e) Automatic detection and recovery about faults

f) Easy to scale out

In the paper, the requirements above and respective solutions are analyzed through introducing a PortLand system for data center networks, Google and Facebook's solution in their data center networks, and Microsoft's testbed, CrystalNet.

2. Portland

In this paper, the authors state the current situation of data centers and then introduce the design and test-based evaluation of PortLand, a set of Ethernet-compatible routings, forwarding, and address resolution protocols specifically tailored for data center deployments [2].
2.1 Design of PortLand
The objective of PortLand system is to provide efficient forwarding, scalable layer 2 routings, and quick detection and recovery for faults. According to observation from data centers environments, the base of multi-rooted network topology is relatively stationary and unlikely to evolve quickly [2]. The architecture of PortLand involves six compounds.

2.1.1 Fabric manager:
The fabric manager is a highly centralized controller running on a machine to assist with ARPs, multicast, detection and recovery for faults, and so on. Restriction of the number of centralized knowledge and limitation of soft state are capable of eliminating the demand for human configuration in the network——number of switches, location, and identifier. The fabric manager is similar to the control center of the data center using PortLand. With the fabric manager, PortLand utilizes the resource with scalability and higher efficiency in many aspects such as more convenient ARP request procedure, more logical address for communication, and insurance for fewer lost packets during VM migration. Moreover, the fabric manager has several duplicates as backups without strict consistency among them because the manager maintains no hard state.

2.1.2 Hierarchical Pseudo MAC addresses:
The PMAC address is the basis for efficient forwarding, routing, and efficient visual machine migration.

PortLand allocates each end host a distinct PMAC address. The form of the 48-bits PMAC address distributed by edge switches that each host connects is pod.position.port.vmid: pod (16 bits) represents the unique pod number given by the edge switches; position (8 bits) shows the position of the host in the pod; port (8 bits) is the port number in the view of local switches; vmid (16 bits) is used for multiplex communication among virtual machines on the same physical machine (i.e. host). Because the PMAC address is as long as the MAC address and the edge switches convert the MAC address into the PMAC address, hosts sending ARP requests will receive the destinations’ PMAC addresses and have the illusion that what they get are the MAC addresses. With rewriting between the PMAC address and the actual MAC address, PortLand simplifies the routing tables. As long as the each part of PMAC address is given, the switches in three stages can forward with a simple check of the PMAC address in the header of each frame.
When an edge switch detects a new MAC address, this edge switch creates a registration in its local PMAC table which maps the host’s MAC address and IP address to its PMAC address. The edge switch then informs the fabric manager about this registration so that the fabric manager can respond to the corresponding ARP requests. The whole procedure is shown in Figure 2.

2.1.3 Proxy ARP:
Originally, hosts in data centers attain addresses by broadcasts. In PortLand, proxy ARP is used to dramatically reduce broadcast traffic.

When edge switches receive ARP requests for IP addresses corresponding to MAC addresses, they send these requests to the fabric manager. Subsequently, the fabric manager checks its PMAC table to find if any registration is available for the target IP addresses. If the fabric manager succeeds, it returns the PMAC addresses to the edge switches and then the edge switches generate ARP replies to the hosts sending the ARP requests; Otherwise, the fabric manager broadcasts requests to all end hosts and the requests are sent to a core switch which then dispenses the requests to all pods to retrieve corresponding PMAC addresses. The target hosts reply with their MAC addresses, which are rewritten by the edge switches into PMAC addresses before sent to both the requesting hosts and the fabric manager. The procedure is shown in Figure 3. In this manner, once the PMAC table in the fabric manager includes a previously unknown PMAC address related to an IP address after one broadcast, any host which requests the same MAC address mapping will receive a reply from the fabric manager directly, reducing the data center’s load of broadcast caused by ARP requests.

Proxy ARP also plays an important role in VM migration, which occurs frequently in data centers. To achieve the goal that a TCP connection cannot be broken during migration, the migration cannot occur at layer 3. For this, PortLand’s solution is the assistance of the fabric manager. After succeeding in migrating to a new physical machine, the VM informs the fabric manager with its new IP address as well as its MAC address. The problem is that other hosts which are communicating with the migrated VM do not update their ARP cache and thus still send packets with outdated PMAC address. The fabric manager will forward a message about the invalidation to the switch which is connected to the previous VM host. The message allows the edge switch to capture the subsequent packets which contain invalidated PMAC address. Also, the switch forwards a unicast ARP to update the hosts, which sent packets with the invalidated PMAC address, with the migrating VM’s new PMAC address. The invalidating switch can also forward the packets to the actual destination optionally. In this manner, during the time of migrating, although the migrating VM is unable to communicate with others, the switch can prevent packets from losing; Moreover, the unicast message forwarded to hosts communicating with the migrating VM enable them to attain the new PMAC address before its ARP cache entry times out.
2.1.4 Distributed Location Discovery:

The switches of PortLand have to have their positions in the entire structure of the data center to work efficiently, for instance, to use PMAC addresses base on location. Although the switches’ positions are relatively static in a data center, a location discovery protocol (LDP) is used to complete discovery about the location without human configuration.

The switches send a Location Discovery Message (LDM) for an adaptable interval to all ports. LDMs contain some information about the switches. The key to location discovery is that the edge switches get LDMs only from the ports connecting to the aggregation switches because end hosts do not generate LDMs [2]. The edge switches can know their location first and realize that the port receiving the LDMs is upward-facing ports connecting to the aggregation switches. They incorporate this knowledge into additional LDMs. The switches receiving LDMs sent from the edge switches are aggregation switches and corresponding ports are downward-facing ports to the edge switches. At last, if the switches that have not yet recognized their location verify that all active ports are linked with aggregation switches, these switches are able to realize that they are the core switches. This election mechanism is useful because, in a given data center, PortLand system can know their local location with LDMs rather than manual configuration, which reduces the rate of human errors which are prevalent in data centers. Knowing their local locations, all switches can easily achieve routing with PMAC addresses.

The unique pod number, which is an ID given by the fabric manager, is the same for switches in the same pod. This pod number is a component of the PMAC address (pod.position.port.vmid), which helps end host achieve an easier forwarding.

2.1.5 Forwarding:

When switches recognize their relative location with LDP, they can populate their forwarding tables by utilizing updates from neighbors. The PMAC addresses are critical to forwarding responsibilities. The core switches are able to easily and correctly forward by simply inspecting the bits in the destination’s PMAC address in the header to get the pod number. Likewise, after attaining all connected edge switches’ position numbers, the aggregation switches are able to determine if the packet is sent to the host in the same pod or a different one by comparing the pod number contained in the PMAC address with its pod number. If the destination is in the same number, the packets will be forwarded through the port which corresponds to the location in the destination’s PMAC address; else, the packets will be sent to any connected core switch depending on flow balancing techniques.

2.1.6 Fault Tolerance:

LDP exchanges can also serve to fault detection and action. The key to detecting failures in data centers is the assistance from the fault matrix of the fabric manager and the assumption that missing LMDs for a period indicates failures. In PortLand, both unicast and multicast can be detected and recovered.

2.2 Implementation and Evaluation

In this paper, the authors use a platform appropriate for the layout in Figure 1 to evaluate PortLand system. Several experiments are run on the platform.

2.2.1 Convergence Time With Increasing Faults:

As figure 4 shows, the average convergence time for UDP required to reestablish communication for 20 runs begins for 1 failure at 65ms and then increases slowly as failures increase.

2.2.2 TCP Convergence:

Figure 5 shows that the average TCP convergence flows spend more time than the one for UDP since transportation using TCP loses a whole window worth of data and the retransmission timer causes the segments to be reset.
2.2.3 Multicast Convergence:
The authors also measure the convergence time needed to employ a new core when the current core of a multicast group loses connectivity to others. As Figure 6 shows, the PortLand system spends 110ms detecting the failures and informing the fabric manager which uses fault matrix to reconfigure appropriate switch forwarding tables.

Fig. 4. Convergence with Increasing Faults [2]

Fig. 5. TCP Convergence [2]

Fig. 6. Multicast Convergence [2]
2.2.4 Scalability:
Although the authors do not have access to a testbed at scale, they still use existing measurements to estimate the requirements for a much larger DCN network and test the scalability of the fabric manager in PortLand design. In a data center with 27648 hosts which has a transmitting speed of 100 ARPs per second, which is in an extreme condition, the fabric manager, as Figure 7 shows, has to handle a 376Mbits/s of control traffic under the situation with the absence of a local ARP cache for usually a 60-second timeout [2]. Additionally, the time required to process each request is approximately 25 µs per request in the implementation without optimizing. To deal with the massive ARP requests under extreme condition, the required number of independent cores is 70, as Figure 8 shows. Although it exceeds the capacity of a single machine, it is practicable to employ the fabric manager in several machines.

Back to 2009 when the paper was published, there was not effect software to simulate data centers. Fortunately, in 2017, Microsoft created a software, CrystalNet [4], which can simulate data center at a relatively large scale and also costs relatively low. CrystalNet may be a suitable testbed for PortLand system at a large scale.

2.2.5 VM Migration:
The ability to support virtual machine migration was measured. As Figure 9 shows, it takes approximately 32 seconds to recover to full speed. Moreover, at approximately 5 seconds in the experiment, the throughput of the TCP flow begins to drop due to the beginning of the migration of a VM.
2.3 Achievement
PortLand achieves several critical goals for the high-efficiency data centers: PortLand migrates VMs with proxy-based ARPs at layer 2 which does not destruct existing TCP connections and also prevents the packets from losing; The human configuration can be minimized by Location Discovery Protocol which can enable all switches to learn their local location; Extremely complex routing table can be avoided by highly hierarchical PMAC addresses and corresponding rewriting; An additional protocol can avoid loop routing in fat-tree topology; The algorithm coupled with the fabric manager using the fault matrix can detect faults and recover them quickly. Moreover, the availability is important as well. The authors design PortLand over a fat tree structure and also the fat tree is a traditional multi-rooted topology in data centers. Therefore, PortLand described above can be employed in the most data centers.

3. Jupiter Rising
This paper [5] presents the evolution of Google data center from the four-cluster-post-network to Jupiter. While on one hand, the process saves a huge amount of money for Google on purchasing the materials and provides the data center with highly scale-able bandwidth, better yet, the project highlights an revolutionary and innovational concept: scale out instead of scale up.

3.1 Four-post-cluster-network
It was called “Four-post-cluster-network” because of the four cluster routers. In this original Google data center network, the network scale is seriously limited. As could be noticed from the picture, there are four clusters on the top. These four clusters compose a wall, stopping Google from getting more bandwidth. At that time, Google network scale is largely limited. To do any upgrade, Google could only purchase the next biggest four clusters, and replace the old four with these new ones, then reconnect the networks. The result is much money spent regularly buying new clusters and the scale limitation still exists due to the latest version of the cluster in the market.

Fig. 9. VM Migration [2]

Fig. 10. Four-post-cluster-network [5]
3.1.1 Scale out instead of scale up
In order to break the limitation of bandwidth, Google started its Firehose 1.0 (or FH1.0) to increase sectional bandwidth to more servers. The goal is to flexibly increase the bandwidth without spending as much money. The team applied Clos Topology and usage of Merchant Silicon to the model.

3.2 Important Concepts

3.2.1 Clos Topology
A clos topology is made up of leaf and spines layer. In clos topology, each servers are interconnected to each switches, and each switches are interconnected to every spine. Thus, in such cases, there’s no all lead nor all spine connections. Clos topology as a result, it is ideal for great fault tolerance and large scale of data center networks. The broadcast domain is minimized by Clos Topology which then makes it more stable. The most important feature here is that Clos Topology is recursive, which means it could provide almost infinite scale by just adding more blocks and spines with more links in between. Drawbacks, however, also exist. Because Clos Topology interconnects Top of Racks (ToR) and Spines with at least two radix for each pair, the routing complexity then increases.

3.2.2 Merchant Silicon
The difference, between merchant silicon to the commonly used commercial switch components that target small-volume, large feature sets, and high reliability, is that merchant silicon has lower price and off the shelf characteristic. In order to meet the bandwidth demands with regularly upgrading the network fabrics, merchant silicon is a wise choice considering increasing bandwidth capacity cost-effectively. Though a common drawback of merchant silicon is that merchant silicon is not feature rich, yet it’s enough for the construction and operation of the data center that Google seeks.

3.3 Failure of Firehose1.0
One drawback topology has is the low radix number of ToR switch. For example, if one of the left uplink of a source ToR and one of the right uplink of a destination ToR failed, machines on these racks couldn’t communicate with each other. Learning from the failure, Googles then made the decision to integrate that switching fabric with adding one custom PCI board.
3.4 Firehose 1.1

Firehose 1.1 (FH1.1) adapts the same network card from FH1.0. However, Google made the network cards into a conventional line form factor, so they could be complied into a chassis. The line-cards are a different form factor from the Firehose 1.0, there’s an “out-of-band” Control Plane Network (CPN), means the chassis has its own control card, which then makes the network more stable than before. And because different layers are interconnected with each groups of two ToR switches, so the work assigned to each switch in Firehose 1.0 was then reduced in Firehose 1.1.

3.4.1 Wire connection challenge

The routing complexity of the copper wire in FH1.1 is very high. The building, testing, and deploying the network require huge amount of manual labor at the risk of error. The main problem is the short length of the copper wire that was applied in the connection. The copper wire that Google uses is CX4 copper cable which has a 14 meters restriction and require very careful placement. Thus, the solution Google came up with is to invent its own cable, a fiber that has enough length. The new cable has a length of 100 meters.

3.4.2 Application of FH1.1

Deploying an unproven new network technology, Google built the FH1.1 next to the old Four-post cluster routers, and the Top of Racks would be connected to both networks at the same time. In such case, they let the Top of Rack forward the default traffic to the four-post cluster, those connectivity to the outer clusters and data centers, and all those traffic then would follow the proper path along the bag-on-the-side network.

And eventually, the experiment turned out to be workable, that google data centers enjoy the more bandwidth along with low-cost. iii) Cabling challenge The main drawback of the Firehose 1.1 is the cabling. So Google replaced the old merchant silicon chips with the most updated merchant silicon 16 bites 10 gigs and arranged them in the line pods. And the chassis then has a backplane, so they could build a non-blocking switch with 128 bites 10 gig ports. And the bulky cables are then gotten rid of, what left are the fibers. As you can see in the picture, the linkage of the four chassis house are all packaged with fiber.

This picture shows how they started the connection of the cluster fabric inside the cluster networking. As you can see, the complexity of the fiber cables in the top picture is very high, and that is only in one watchtower. A concept implemented here is “cable bundling”. Google takes a bund of physically colocated group of chassis and bundling all those fibers between these groups of chassis. The cable bundling not only makes the routing more neat and easy, also saved the cost nearly 40

![Fig. 13. Reducing deployment complexity by bundling cables](image)

3.5 Jupiter rising

Finally, Google came to its famous Jupiter. Next generation of merchant silicon 24 bte 10 gig allow google to build even more and bigger non-blocking switch to seven chassis 228 by 10 gigs ports, and
importantly, they introduced the same silicon on top of racks called Pluto ToR that allows host to burst at 10 gigabytes per second each for the first time in the data center. Based on that, Jupiter network introduce 40 gigs silicon which could be arranged in higher building blocks and can finally conform to the flow-based architectures of aggregation blocks interconnected by the spine blocks that can be scaled out. This merchant silicon is a high bandwidth as well as its higher radix to allow them to scale out such network to an entire building with aggregated bisection bandwidth of 1.3 petabytes per second.

4. Reliability of the Data Center

It is known to all that it is important for large scale Internet companies to possess a reliable network infrastructure, for the network collapse may lead to chaos. To build and operate highly-available web services, the network structure needs to be capable of tolerating and recovering from network failures. So, engineers and operators need to have a decent knowledge of network reliability, as well as its implications on the software systems in data centers [6]. The paper represents a wide and deep study of the reliability of the data center network. They collected data on Facebook’s production network infrastructure and use it as a data source. They study the characteristics of the reliability of both outer and inner data center networks, the influences of network reliability have on the software systems, and the evolution of the reliability characteristics [6]. In the section, implications of network reliability on the design, implementation, and operation of large-scale data center systems and the influence of reliability on highly-available web services are discussed. The researchers try to lay a foundation for people to understand the meanings of the reliability of network infrastructure on a large scale, and help people inspire new ways to solve the problems of the network.

4.1 Facebook’s network infrastructure

This figure demonstrates the structure of Facebook’s network. In the picture, we can see three different regions. The left one (Region A) is called cluster network, the right one (Region B) is fabric network, and the one in the middle is WAN backbone. Technically, the network within data centers—cluster network and fabric network—is called, well obviously, the intra data center network; it is also referred as the WAN backbone network for the inter data center network which is between data centers. The design for cluster network is older, while the design of the fabric network is newer.

4.1.1 Cluster network:

A cluster network is one of the kinds of network infrastructure, where two or more computing devices working together for one computing purpose. In cluster network, a cluster is a fundamental unit for network deployment. Each cluster consists of four cluster switches (CSWs), and each CSW sums up the physically contiguous rack switches (RSWs). Then, a cluster switch aggregator (CSA) gathers...
CSWs and core network devices, also being referred as core devices, aggregate CSAs, where inter data center traffic flows through [6].

Though the cluster network remains in use, there are two primary limitations:

a) Vendor devices limit data center scalability [6]. Cluster switches aggregate with other switches physically, so if we want to connect more devices, we need to wait for vendors to produce larger switches, which limits the development of the size of data centers.

b) Proprietary software is challenging to maintain and customize [6]. Once they have been implemented, proprietary switches have to be recovered in-place. Humans need to power recycle the device when a device goes wrong, and it takes a long time. As a result, there are fewer switches running, leading to more traffic in the remaining switches, resulting in more traffic congestion in the network.

4.1.2 Fabric network:
To solve the problems, people use a new network design called fabric network.

In Figure 17 we can see, a pod is the prime unit of the network implementation in a fabric network. Compared to a cluster, where RSWs are physically contiguous, RSWs in a pod without any physical limitations. Each RSW links with four fabric switches (FSWs), which is as same as the cluster network, for the 1 in 4 ratios of RSWs to FSWs have connectivity benefits. the number of FSWs which is aggregated by Spine switches (SSWs) depends on the software. Besides, each SSW links with a set of edge switches (ESWs); and core devices connect ESWs between data centers [6].

Fabric network is quite different from cluster network:

a) Simple, custom switches. Fabric devices do not need to spend a lot of time waiting for the vendors to produce larger switches, for their sheer, commodity chips and eschew proprietary firmware and software.

b) Fungible resources. Unlike cluster network, fabric devices do not link to a strict physical hierarchy. Resources could change dynamically depending on the demand of network for bandwidth and reliability.

c) Automated repair mechanisms. In fabric network, some kinds of failures on devices can be fixed automatically by the software.

d) Stacked devices. A virtual device with a higher bandwidth can be created by stacking the same type of fabric device into the same rack. Compared to the port density in proprietary network devices, that of in fabric networks scales faster by the stacking process [6].

4.1.3 WAN backbone:
The WAN backbone design used by Facebook is traditional, for it consists of the backbone routers in every edge node. Located in edge nodes, backbone routers (BBRs) provide routers for cluster networks and fabric networks to transmit data across the WAN backbone and Internet.

Facebook’s Wide Area Network (WAN) backbone is comprised of edge nodes connected by fiber links, formed by optical circuits made of optical segments, which are optical fibers that connect edge nodes [6]. It’s important to keep fiber links connected to those software systems implemented in multiple data centers. Otherwise, serious network incidents will happen.

4.2 Intra data center ability
For this part, we make a deep study of the reliability of data center networks. In the study, researchers learn about network failures by three perspectives: root causes, device type, and network design. Also, they categorize network failures on the basis of the network structure. The study analyzes root causes, device reliability, incident rate, and distribution, network design, and incident severity.

The data source they use is collected from the SEVs ranging between 2011 and 2018. SEV’s full name is Site Event, which is written by Facebook’s engineers about network incidents, containing the root cause and its effect on software systems, and how to take precautions.
Table 1 Common Root Causes of Intra Data Center Network Incidents at Facebook From 2011 to 2018 [6]

<table>
<thead>
<tr>
<th>Category</th>
<th>Fraction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance</td>
<td>17%</td>
<td>Routine maintenance (for example, upgrading the software and firmware of network devices).</td>
</tr>
<tr>
<td>Hardware</td>
<td>13%</td>
<td>Failing devices (for example, faulty memory modules, processors, and ports).</td>
</tr>
<tr>
<td>Misconfiguration</td>
<td>13%</td>
<td>Incorrect or unintended configurations (for example, routing rules blocking production traffic).</td>
</tr>
<tr>
<td>Bug</td>
<td>12%</td>
<td>Logical errors in network device software or firmware.</td>
</tr>
<tr>
<td>Accidents</td>
<td>11%</td>
<td>Unintended actions (for example, disconnecting or power cycling the wrong network device).</td>
</tr>
<tr>
<td>Capacity planning</td>
<td>5%</td>
<td>High load due to insufficient capacity planning.</td>
</tr>
<tr>
<td>Undermined</td>
<td>29%</td>
<td>Inconclusive root cause.</td>
</tr>
</tbody>
</table>

4.2.1 Root causes:
From Table 1, we can observe that the undetermined incidents account for the largest proportion. Excluding undermined incidents, maintenance failures account for the most documented root causes. Then, the possibility of hardware failures’ occurrence is equal to that of misconfiguration failures. Bugs and accidents have similar rates. Figure 15 shows the effect that each root cause has on different kinds of devices in the network. Most network device types are influenced by major root cause categories, such as hardware, maintenance, misconfiguration, etc. Also, a portion of the root cause categories is characterized unequally between devices, such as maintenance.

From the observations above, researchers conclude for network incidents that are documented, maintenance failures occupy the majority. Furthermore, the number of human errors, such as bugs, is twice the number of failures caused by hardware.

4.2.2 Incident rate and distribution:
Generally speaking, the reliability of every interconnected network device is the determining factor of the overall reliability. Researchers define the incident rate as the number of incidents resulting from the type of network device per active device in the network of that type. In Figure 16, there was an inflection point in 2015, when the fabric networks got into deployed.
Fabric network devices have lower incident rates than cluster network devices. Figure 17 shows how incidents resulting from each type of network device distribute on the order of years. It is obvious that after the fabric network got into deployed, devices in fabric networks did not increase so much. Instead, fabric network failures take a very small proportion. This may suggest that fabric network has better fault tolerance.

In this part, we conclude that fabric-based data center designs may have better fault tolerance.

### 4.2.3 Incident Severity:

There is a SEV level that represents an incident’s high watermark in Facebook’s network incidents. Incidents are classified into 3 severity levels. SEV3 is the lowest while SEV1 is the highest.

Figure 18 presents the distribution of the SEV types in 2017 among devices. As is vividly shown, core devices account for the largest proportion of SEVs and the severity of cluster network is higher than that of fabric network devices.

<table>
<thead>
<tr>
<th>Table 2 SEV Levels and Incident Examples [6]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level</strong></td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>SEV3</td>
</tr>
<tr>
<td>SEV2</td>
</tr>
<tr>
<td>SEV1</td>
</tr>
</tbody>
</table>
4.2.4 Network design:
Data center network design is of significance in network reliability. Figure 19 shows an inflection point occurring in 2015, which is similar to the observation in the former part. The adoption of fabric networks contributes to the turnaround. Similarly, there is also an inflection point in Figure 20, which presents that changes happening in the fraction of network incidents when fabric network replaced cluster network. As is vividly shown in the figure, the number of cluster network incidents grew rapidly until 2015, the time when the populations of cluster network devices began to decrease. Besides, in 2017, incidents for cluster network devices increased to almost twice as many as that of fabric network devices.

Fig. 19. The number of network SEVs overtime normalized to the number of deployed network devices [6]

Fig. 20. Population breakdown by network device type [6]
In this part, we conclude incidents of cluster networks are much more than that of fabric networks. What’s more, fabric networks are more reliable than cluster networks for their simple, custom switches and automated repair software, adapting to tolerate more errors [6].

4.2.5 Device Reliability:

To make an analysis of the reliability of Facebook’s data center network devices, researchers take notes of when the incident starts and when it is solved, and then work out the mean time between incidents (MTBI) and 75th percentile (p75) incident resolution time (p75IRT).

They choose the three methods (MTBI, p75, and p75IRT) because engineers in Facebook record resolution time, which contains some extra time, leading to inaccurate results. To avoid errors, they examine p75IRT.

Figure 22 diagrams MTBI for each switch type by year[6].

From the figure, firstly, from 2011 to 2017, MTBI did not fluctuate too much except CSAs. This is due to operational improvements in 2015, which increased CSA MTBI. Secondly, comparing MTBI to switch type population size in 2017, we make assumptions that larger population sizes may lead to longer MTBIs. The reasons lie in the engineers at Facebook, who paid more attention to deploying techniques instead of devices.

Then, researchers measure the meantime of the beginning and the end for p75IRT. The results show that p75IRT increased at a similar speed across different device types. Thus, we could make assumptions that p75IRT has a positive correlation with the number of devices.

In summary, as regard to device reliability, incident rates vary by three aspects in different device types and larger networks may lead to a longer time for incident resolution.
4.3 Inter data center reliability

For this part, we analyze link failures and edge node failures to research the reliability of backbone networks. In the analysis, researchers measure fiber link meantime between failures (MTBF) and mean time to repair (MTTR) based on the repair letters parsed and stored in the database in Facebook.

4.3.1 Edge Node Reliability:
We find the reasons for most edge nodes failing infrequently are unreliable links in fiber vendors. By observing, we point out that the figure looks like an exponential function, so we model MTBFEdge(p) (an exponential function of the percentage of edge nodes) and the fitting result is good.

The same methodology is used in the analysis of mean time to repair (MTTR). We find the time spent on edge node recovery is less than the time between failures, for edge nodes have large numbers of links and fiber vendors, so that they could work to repair link failures immediately. As the same as above, we use the same exponential model and the fitting result is good.

4.3.2 Link Reliability by Fiber Vendor:
Again, researchers make an analysis of the MTBF and MTTR based on the recorded time the operating links fail or recover and model MTBFvendor(p) as an exponential function of the percentage of vendors.

Fig. 23. MTBF as a function of percentage of edge nodes connecting Facebook data centers with that MTBF or lower [6]

Fig. 24. MTBF as a function of percentage of fiber vendors with that MTBF or lower [6]
We find that in the majority of cases, link failure only happens occasionally. This is because the links of regular maintenance and monitoring fail less than once every 2326h, or 3.2 months [6]. It can be seen in the figure that the variance of fiber vendors in MTTR is high. We infer that this is because many fiber vendors choose to operate in areas where the links are easier to repair. Again, we model MTTRedge(p) as an exponential function of the percentage of vendors, and the fitting result is very good. In summary, fiber vendors operate unequally.

4.3.3 Edge Node Reliability by Geography:
The researchers make an analysis of the reliability of edge nodes in the respect of geographic location (the continents they are located in). Table 3 above shows the distribution of the Facebook network’s edge node across continent. Most of the edge nodes are located in North America, next is Europe. Africa and Australia have the fewest edge nodes. Researchers state that edge node reliability in Africa is of great importance, on account of the low number of edge nodes in Africa and the location of Africa (between Europe and Asia). The MTBFs of edge nodes vary, and the standard deviation across continents is 1333h.

By analyzing MTTR, we find that edge nodes in Africa spend the most time on recovery, partly because of their submarine, while Australia’s edge nodes spend the least time because they are located in big cities.

In summary, the edge node failure rate varies by locality, and edge nodes typically recover in 1 day across continents [6].

4.4 Conclusion
In the paper, we do research on the data collected from Facebook’s data centers and make a lot of interesting observations, such as human errors are twice as many as hardware errors; fabric networks
tend to have fewer incidents than cluster networks; larger networks are likely to take longer incident resolution times; time to failure and time to repair seem to follow the law of exponential functions; backbone edge nodes approximately fail once per month and recover once per hour; edge failure rate varies by locality and so on.

Through analysis, we learn about the characteristics of the reliability of intra and inter data center networks, and the evolution of it. Particularly, we make discussions on the implications of network reliability on the design, implementation, and operation of large-scale data center systems and the influence of reliability on highly-available web services.

The study is meant to help people better understand the reliability of data center networks and come up with more good ideas to improve it in large scale data centers.

5. Crystalnet

This passage introduces the CrystalNet which is developed and used by Microsoft to detect bugs in network operations by simulating the network. It replaces traditional emulation applications like Batfish, a large scale emulator which cannot find bugs in routing software, issues with different vendor implementation, or human error, which could all be solved by CrystalNet, and MiniNet or GNS3, which can solve some of the problems Batfish could not solve and is more precise but cannot emulate large public cloud networks. Furthermore, operators can use the same software tools and scripts they use to interact with the production network with CrystalNet. CrystalNet is also cost-effective: It costs about 100 dollars per hour to emulate a network of 5000 devices, which is comparatively low for emulating such a large network. What’s more, CrystalNet is able to accommodate a wide range of router software images (E.g. virtual machines connected to the network) from different vendors. Finally, it is very hard for a network emulation application to emulate the network outside the boundary for it is impossible to mock up the whole internet. However, CrystalNet could mock up the external network [4].

5.1 Common bugs in the network

Microsoft engineers developed CrystalNet to detect bugs, which can be categorized into software bugs, configuration bugs, and human errors, in their product.

<table>
<thead>
<tr>
<th>Root Cause</th>
<th>Proportion</th>
<th>Examples</th>
<th>CrystalNet Coverage</th>
<th>Verification Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software bugs</td>
<td>36%</td>
<td>Bugs in routers, middleboxes, management</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Config. Bugs</td>
<td>27%</td>
<td>Wrong ACL policies, traffic black holes, route leaking</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Human Errors</td>
<td>6%</td>
<td>Mis-typing, unexpected design flaws</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Hardware Failures</td>
<td>29%</td>
<td>ASIC driver failures, silent packet drops, fiber cuts, power failures</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Unidentified</td>
<td>2%</td>
<td>Transient failures</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Software bugs account for 36% of the total error reported; usually, this type of bugs include bugs in routers and management tools. Configuration bugs take up 27% of the total bugs; it usually includes bugs like traffic black hole or route leaking. Hardware failures have 29% of the total bugs reported; this type of bug has a wide range of reasons causing it, like power failures, cache overflow, and so on. Human errors take up a relatively small portion: only 6%; this type of bug is usually caused by design flaws or mistyping. And the 2% error bugs that are leftover are unidentified. For all of the types of bugs mentioned above, traditional network emulation applications or verification
applications can only detect the configuration bugs. CrystalNet, on the other hand, can detect all of the types of bugs identified except for hardware failures.

5.1.1 Software Bugs:

In the figure above (Fig. 26), R1 has 2 IP prefixes, P1 and P2, and R6, and R7 will receive the announcement of 2 IP prefixes, through R2, R3, and R4, R5 respectively, and aggregate the IP prefixes to R8 for P3. However, R6 and R7 are from two different vendors, and use different techniques to aggregate the prefix for R8 to transmit the announcement. In this case, R8 will always select the path given by R7, because it perceives R7 as a faster route, causing server imbalance. And CrystalNet can detect the server imbalance and ping it out for the operators to fix the bug.

5.1.2 Configuration Bugs:

Network configuration incorporates a lot of things like CPU load, security, and access control. So it is quite intricate when taken together and causes a lot of problems like overlapping of IP assignments or incorrect Anonymous System (AS) numbers. Microsoft has already implemented programs to reduce those types of error before the creation of CrystalNet, but CrystalNet could further reduce the configuration bugs.

5.1.3 Human Error:

With CrystalNet, operators could have an environment too, for example, test their implementation plans, just like the debug mode in a programming compiler.

5.2 Design Goals

The followings are the design goals for the CrystalNet.

5.2.1 Easy to scale out:
To understand this feature, an example would help you understand it: If an operator wants to double the network being emulated, CrystalNet would simply achieve this by allocating twice the resources (Example 1).

To achieve this design goal, engineers of CrystalNet use an overlay network (showed in Fig.27. The overlay network is built on top of the Virtual Machine’s (VM) cluster. So when some modifications are made in the lower layer, the modifications will also be applied to the upper layer. In Example 1’s case, if the operator expands the range of the emulation target two times, the upper layer, the emulation network would also double, taking up twice the resources. [7]

5.2.2 Ability to transparently mock-up physical networks:

After the CrystalNet is able to transparently mock up the physical networks, devices connected to the virtual network can be accessed easily. [8]

Since the topology of the overlay network, PhysNet in this case, is identical to the emulation target, the emulated devices give the same routing information, just like in the real networks. In the emulation network, each virtual machine work independently in the network and orchestrator, just like the one in an orchestra, direct the whole "band" to work tidily and can easily detect and restart the failed virtual machines.

To mock-up the physical networks, a programming interface allows operators to make changes to the emulation network, just like that of in the real physical networks.

Fig. 28. A typical network update validation workflow. CrystalNet APIs cover the parts in blue and bold. The rest of the workflow is operator-specific.[4]

Fig.28 shows the whole process, which allows the operator to implement operations on the emulation network, using the CrystalNet application programming interface (API). Table V describes the specific functions running when the API is executing the general functions. The provision functions are the step for the API to set up and getting prepared. The following control function is for the operators to give operation (usually the modification the operator wants to apply to the emulation network) to the CrystalNet. And the monitor function shows the result and details after running the emulation. If the operator is satisfied with the emulation result, he or she will move on, else repeat the whole process of the control function to the monitor function until that operator is satisfied with the outcome.

However, there are several problems when implementing the idea of mocking up the Physical Network. First, devices connected to the network have different software systems, and CrystalNet must support and run on all of those devices. Second, since each device has a different software system, the API has to be designed to accommodate every system, which is nearly impossible. Thirdly, operating system-level virtualization, also referred to as containerization, is essential for CrystalNet. It has the ability to achieve software’s live migration, which acts like the role of a container: put the software into the "container” and move it into another operating system where the software could work just like before. However, this containerization must boot with the interface implemented into
each device, while CrystalNet’s virtual interface can only be implemented after the container is
booted.

Table 5 Selected Crystalnet Apis.[4]

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Provision Function</strong></td>
<td></td>
</tr>
<tr>
<td>Prepare</td>
<td>Gather information for Mockup. Spawn VMs.</td>
</tr>
<tr>
<td>Mockup</td>
<td>Create the emulation based on Prepare output.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clean up everything on VMs.</td>
</tr>
<tr>
<td>Destroy</td>
<td>Erase all Prepare output. Including the VMs.</td>
</tr>
<tr>
<td><strong>Control Functions</strong></td>
<td></td>
</tr>
<tr>
<td>Reload</td>
<td>Reboot devices with specified software versions and configurations.</td>
</tr>
<tr>
<td>Connect</td>
<td>Connect two interfaces.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnect two interfaces.</td>
</tr>
<tr>
<td>InjectPackets</td>
<td>Inject packets with a specified header from a specified device &amp; port, at given frequency in given amount of time</td>
</tr>
<tr>
<td><strong>Monitor Functions</strong></td>
<td></td>
</tr>
<tr>
<td>PullStates</td>
<td>Pull common states from the device software, e.g. FIB, RIB, CPU and memory usage, etc.</td>
</tr>
<tr>
<td>PullConfig</td>
<td>Back up the current configuration for rollback.</td>
</tr>
<tr>
<td>PullPackets</td>
<td>Pull the packet traces to local, and (optional) compute packet paths and counters (optional) clean traces after pulling.</td>
</tr>
</tbody>
</table>

**Management plane: complementary to Control and Monitor**

| IP Access | Enable existing tools to send commands or pull outputs. This is not a typical API. |

Fig. 29. PhyNet containers in CrystalNet decouple the interfaces’ management and facility tools from the device software.[4]

Engineers separated the mocked physical network from the devices being tested (the details of separating the Physical Network are explained in the following section). And APIs are implemented in the mocked Physical Network Container to address the first problem since the mocked Physical Network will be able to accommodate different software systems; the second problem is also solved as a result. Then the actual device is booted with corresponding share network namespace, so when started, the Physical interface is already configured. There are additional benefits that come with separating the physical network from the devices: even when the devices crash, the virtual interface still exists and can reboot the devices.
5.2.3 The Ability to Mock-up External Networks:

Every network emulator has a boundary since it is impossible to emulate the whole Internet. But the engineers have this goal that CrystalNet should be able to achieve high fidelity emulation of the network that is connected to devices outside the emulation boundary.

Fig. 30. An unsafe static boundary to emulate T1-4 and L1-4.[4]

It is evident that to mock up the external networks, the first problem to solve is the emulation boundary problem. The boundary stops where emulated devices are connected to the wide-area Internet. To ensure the correctness of the network emulation, the reaction from outside the boundary is indispensable. [9] The key to solving this requirement of the external reaction is that most production networks have protocols to limit the range of impact of route updates, which basically defines where the boundary is.

After setting up the emulation, it is essential to find a static and safe boundary to ensure the information exchanged in the network to be reliable and the connections are stable. [10] Speaker devices, which are marked in yellow in Fig.30, will not respond to any routing messages transmitted from boundary devices, the ones connecting to external devices marked in blue in the figure.

The boundary showed in Fig.30 is an unsafe one since S1 and S2 do nothing and when they receive the change of IP Prefix in T4 do not forward it to L1, and L2.

Fig. 31. A safe static boundary to emulate T1-4 and L1-4.[4]

Fig.31 is a safe boundary since S1 and S2 are no longer speaker devices, so when the IP Prefix of T4 is announced, every device inside the boundary receives the announcement, thus ensuring a reliable data transmission making the boundary in Fig31 a safe boundary.
5.3 Implementation

There are two phases while implementing the CrystalNet: emulation prepares phase and mock-up phase.

5.3.1 Prepare Phase:

During the preparation phase, CrystalNet is generating topology and configurations. CrystalNet identifies the topology of all devices in the input list and rearranges the format to a form that mock-up could understand. It also estimates the number and types of virtual machines for the preparation of the mock-up physical network.

5.3.2 Mock-Up Phase:

Since mock-up time determines the time and the cost of running CrystalNet, Microsoft uses Linux Bridge because of the fast set up time. CrystalNet groups the virtual machines, each dedicated to just running devices from one vendor, to avoid settings conflict from different vendors’ code. CrystalNet also has a health check and auto-recovery algorithm in case of virtual machine failure.

5.4 Operators and engineer's experience

This section analyzes the users’, operators’, and engineers’ experience after CrystalNet had been deployed for 6 months. The following subsections summarize two cases to demonstrate the effectiveness of CrystalNet.

5.4.1 Case 1: Data Center Upgrade:

In this case, Microsoft is planning to upgrade one of its data centers to improve performance. There are a lot of things connected to that data center which makes the process of upgrading complex and
risky; a mistake would be disastrous: it will cause financial penalties and, more importantly, loss in customer’s confidence. The operators responsible for this upgrade used CrystalNet to test out their plans and software tools. As a result, they found over 50 bugs, which could have caused serious problems, and finished the upgrade perfectly with the help of CrystalNet.

5.4.2 Case 2:

Switch OS Development: In this case, Microsoft engineers wanted to build a validation pipeline for developing a private version of switch OS CTNR-B. They used CrystalNet to emulate the production network and verify that the network behavior is constant. In addition, they found 10 serious bugs in their network.

5.5 Discussion and Drawbacks of CrystalNet

First, CrystalNet is unable to determine the timing and the order of prefix announcements. For example, if the R6 chooses path randomly or based on timing like Fig.26, CrystalNet is unable to know the exact path the data is transmitting, which causes non-deterministic BGP behavior.

Second, although CrystalNet has an effective algorithm for finding small boundaries, its algorithm is not optimized, thus cannot find the smallest boundaries possible. The CrystalNet engineers are still working on it.

Third, CrystalNet is designed for network emulation to test out operator’s ideas and software implementation to real data center networks, like the debug mode in the programming compiler. So it is not suitable for testing of bandwidth or latency, memory leaks, and timing-sensitive bugs. There might be further benefits or drawbacks for operators to explore CrystalNet. The developers of CrystalNet left the testing methodologies for operators while they working with those problems.

6. Conclusion

PortLand is a desirable data center network protocol. In terms of efficiency of data center network, PortLand have several property, such as highly centralized manager, pseudo MAC address, proxy ARP, automatic location discovery, loopfree forwarding, high scalability, and effective fault tolerant routing, which can distinguish it from other data center protocols.

Today, the effort of Google applied into the Firehose project appears to be worthy. The application of Firehose 1.1 provides Google a huge amount of void bandwidth lying inside their data center, while saving hundreds of thousands dollars each year comparing to the former design of four-cluster network.

Unlike previous work, we make a deep study of the influences data center network have on the design, implementation, and operation of large-scale software systems which run highly-available web services. In the paper, we discuss a lot on the root causes of data center network incidents, the incident rate and distribution, the incident severity, the network design, the device reliability, the edge node reliability, the link reliability by fiber vendor and so on.

Section VI synthesizes all of the knowledge previous research paper introduced and put it into real application. Engineers of Microsoft created a Data Center emulator using some of the concepts introduce in previous papers. Although there are little drawbacks, those do not affect the overall availability of CrystalNet on testing data center networks. Overall, CrystalNet is a successful project which not only saves software engineer’s time on testing these software tools, but, more importantly, saved tons of money loss caused by data center malfunctioning and loss in customer’s confidence for Microsoft.

References


