Google – Network Strategies And Architectures

Jonas Miederer
email: jm104@hdm-stuttgart.de
enrolment number: 32723

July 29, 2016

Abstract

In this paper, the key concepts of Google's networks and its components are described and evaluated. In order to receive an impression of what the company actually has to manage, an overview over the size and traffic share is given, although Google does not provide any numbers, thus an exact estimation is not possible. It is shown, that one reason for their fast and reliable services with high availability is the deployment of edge-nodes at peering points, offering advantages to the consumers, the service providers and Google itself. The centralized control over the global backbone WAN is possible by leveraging software-defined networking, which gives Google the ability to manage and monitor the data flow between their sites, manipulate network structures and dynamically deploy rules to the hardware devices. Google combines software-defined networking with traffic engineering in order to define multi-path routes to avoid congestion and keep an even utilization within the network. A closer look at the first and the latest generation of Google's data center networks reveals the change and the constantly ongoing development of the architectures. The shift from a "four-post" cluster to a three-staged Clos architecture has the benefit of scalability, flexibility and a strongly increased bisectional bandwidth. Even though these architectural decisions are well designed, they don't follow Google's ambitions to find the new breakthrough technology, but are more a combination of refined interacting components.

1 Introduction

Google does not just focus on one single business segment, Google is a company offering more than just web searching: It provides web analytics tools, it has its own mailing service, its own office-suite, the video streaming platform *YouTube*, the mobile OS *Android*, *Google Maps*, the web browser *Chrome* and many more. They are involved in numerous business activities, now gathered under a holding company called *"Alphabet"*.

The holding's revenue increased by a factor of four in the last eight years [1], which is why Google belongs to the largest and most important companies worldwide. An internet-based tech-company of this scale requires without doubt a highly reliable, performant, scalable and flexible network in order to provide the services in an optimal way. This is not only important for their internal data center networks, but also for the interconnection of those individual networks, including the edge nodes. In times of growing internet usage driven by the demand for entertainment, communication and connectivity, Google has to keep pace with the development of the internet. In 2020, the global IP traffic will reach more than 194 Exabytes per month, growing by a factor of three from 2015 [6]. The so-called "*Zettabyte Era*" urges Google and other global internet companies to improve their networks, to find new designs and architectures in order to serve and transmit data efficiently and fast and to find new concepts to meet the requirements by the consumers.

The second section of this work provides a short overview over the size of Google and their contribution to the total internet traffic. The third section describes the details of Google's network architectures and methods and is divided into three parts: The first part explains the concepts and benefits of the company's edge-node caching system called *Google Global Cache*. The next part is primarily concerned with the interconnection network *B4*, representing Google's internal global WAN. The focus is on the modern concept *software-defined networking* together with its implementation *OpenFlow*, describing how it works, explaining its advantages and presenting how Google adopted it. The third part explains the principles and concepts behind Google's first and its latest generation of data center networks.

2 Google's size

Google is widely known as one of the largest companies worldwide, providing information, data and services to users all around the globe. An internet company of that size requires without doubt a highly reliable and performant network, capable of transmitting huge amounts of data. Unfortunately, Google is very strict about their data and traffic statistics, they don't provide or publish any information regarding the amount of internal or external traffic. In 2010, Google was estimated to be accountable for about 5.2% of the total internet inter-domain traffic, representing the third largest contributor and the largest non-ISP contributor [19]. Google also exhibited the biggest increase in traffic share, since they migrated videos from a CDN into their own network. Although these numbers give an impression of how big Google really is, they are no longer up to date and don't represent the company's current traffic share anymore. Three years later the same author Craig Labovitz commented on Google's size again, now asserting that the company is accountable for 25% of internet traffic on average [18]. However, he did not state how he came up with this number or how it was measured, so it's just an assumption and was neither confirmed nor denied by Google.

Although Google offers a wide range of services, the most traffic-critical application is the video-platform *YouTube*. The video-demands strongly outpace other less traffic-critical regarding bandwidth. In Europe, more than 28.7% of aggregate peak internet traffic comes from video applications [21], in North America it's even more than half of the total traffic with more than 52% [22]. On both continents, *YouTube* is among the top contributors; in North America they are placed on the second rank after *Netflix*, another video streaming company, in Europe they are on the first place with almost 25% of downstream traffic. These numbers show that *YouTube* is one of the frequented websites, serving large amounts of traffic. In contrast to *Netflix*, *YouTube* is a video sharing platform, so the users cannot only consume videos, but also upload their own files onto the servers. According to Google, users upload 400 hours of video every minute worldwide [3], which makes about 6084 days per year. Unfortunately a comparison between up- and download-rates are not possible, since there exists no such report about the global traffic share of *YouTube*. Nevertheless, *Google* is without doubt one of the largest companies, serving and processing information and data across numerous services, while it plays an important role of everyone's daily life.

3 Google's network architectures and structures

3.1 Google Global Cache

Google offers their services to a variety of countries, acting as a global data and information provider. That means Google must provide a high availability in order to satisfy the user's needs such as looking up information, watching videos or checking the mails at any time. However, there is another important property of networks to be considered: In times of faster and stronger devices, better internet connections and the gaining popularity of (live) video streaming, the response time of requests by the users become more crucial. Generally speaking the latency itself depends on several factors: The material and properties of the links (fibre optic, copper,...), the available bandwidth, the network and server utilization, the computation speed/power of the server as well as the physical distance between client and server. Consequently the user experience improves when the servers are located closer to the consumers, while this approach also leads to higher operational effort and expenses, evoked by the decentralization of the nodes.

Google officially operates 15 core data centers [12], distributed across America, Europe and Asia. Assuming there exists no data center in Africa or Australia, all requests to Google would have to take transatlantic respectively -pacific routes from there, resulting in a very high latency. As already described above, Google's most traffic-consuming service is *YouTube*. In contrast to simple search queries or other text-based information, YouTube delivers large amounts of video-data, not only having impact on down- but also on uplink-rates. It is very cost- and time-consuming to completely transmit the same contents over thousands of kilometers multiple times, especially for viral and popular videos with high click-rates. This inefficiency has disadvantages for the provider (Google) as well as for the consumer. So the solution provided by Google is a large decentralized caching architecture, deployed all over the planet, called ***Google Global Cache (GGC)***.

Google places these caching servers at the edge of their network, as close to the consumer as possible [11]. In the last years there was a major change in network topologies, large companies called "Hyper Giants" or "Over-the-top (OTT) players" constructed their own backbone networks, directly connected to the Internet exchange points (IXP) and peering to the global and national backbone operators [19]. That way, a request from a customer network is sent to the network of the respective Internet service provider, where it finally reaches the Google network by transmission at the exchange point. In conclusion, the closest autonomous system (AS) to the user's network is the ISP's network, where Google actually deploys the GGC-servers.

In order to operate the GGC servers, a peering agreement has to be established between Google and the ISP. Subsequently Google provides the hardware, which is installed by the service provider at their peering point. Figure 1 shows the procedure when requesting a resource from Google, served by a GGC server.

In the first step, the consumer requests content from Google, for example a YouTube-video. If the domain name is not already cached by the client, a DNS-request is sent to the DNS server of the ISP. If the IP address isn't known to the ISP, too, then it is looked up using the Google DNS server (step 2). If the ISP has installed a GGC-server, the Google DNS server recognizes this and returns the respective IP address of the cache server located at the ISP instead of the "original" Google server. The corresponding IP address is then returned to the user as a DNS response (step 3). Now that the consumer obtained the actual IP address, he can start his request from the given server (step 4). Since he received the IP address of the GGC server, there is no direct communication between client and Google server established. If the requested content is not already cached at the GGC node (cache miss), it is transmitted from the originating Google server and cached for future requests (if the content is cachable) (step 5). Finally, the video file is sent from the closely located caching server to the client (step 6).

This approach results in multiple advantages for every involved party:

- **Consumer**: The consumers benefit from the lower latency achieved by the closer location of the server. The most time-consuming part of the content-delivery is the transmission between the ISP-network and Google's network (step 5), which is superfluous with a successful cache hit at the GGC. The elimination of at least three round-trip times (RTTs) leads to faster loading of YouTube videos, map tiles or Android updates and therefore improves the user experience and the satisfaction with the services provided by the ISP as well as Google.

- **ISP:** After installing a GGC node, service providers profit from lower transit costs. That is to say, after a cache hit there are no other networks involved which have to be passed through, potentially incurring transit fees for carrying the "external" traffic. Moreover, the total traffic is reduced, dropping the operational expenses of the service provider. Another key thing to remember is that lower latency leads to satisfied and pleased customers, an essential business factor.

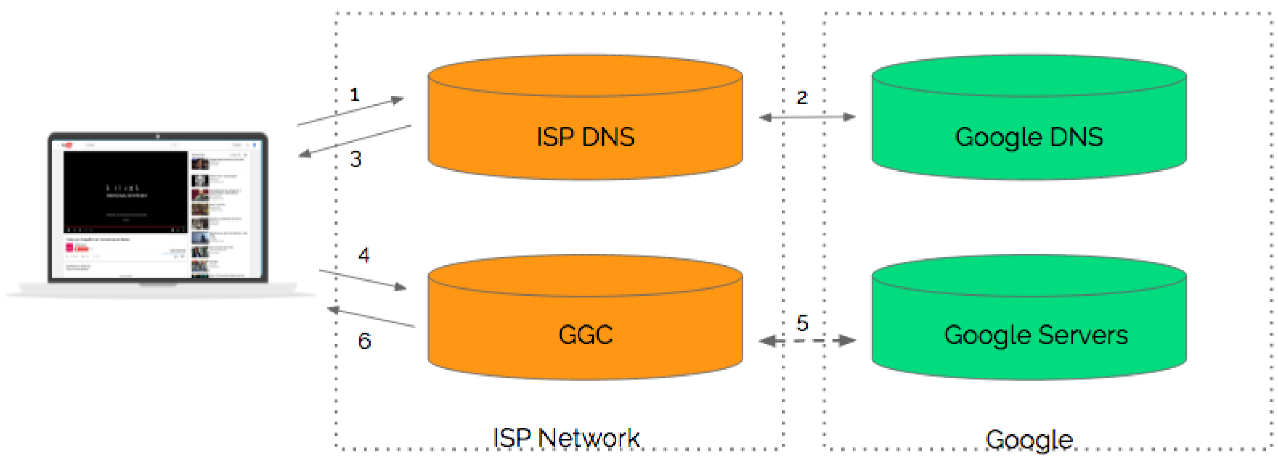
- **Google:** It goes without saying that Google itself profits from their own *Google Global Caches*,too. The main reason for installing these nodes is that Google can heavily decrease their backbone traffic inside their own networks. The company can save on infrastructure, additional load balancing instances and bandwidth by decentralizing the servers. Depending on the consumer behavior, Google can serve 60% - 80% of the total traffic this way [10] and thus most cachable content can be delivered without leaving Google's edge node. The total utilization of internal servers and networks decreases, which has a positive impact on overall performance. By the same token, the GGC architecture provides redundancy: When any cache node fails, the request can be redirected to the next instance, increasing the reliability. Google can profit from the positive user experience as well as the service provider by satisfying the consumers expectations of fast and reliable results in terms of usability.

3.2 B4

3.2.1 Introduction

Google provides their services all around the world, offering highest availability and performance. Besides the global cache system there is another essential architecture responsible for the successful achievement of these goals, called **B4**. From a general point of view, *B4* is Google's main infrastructure. To be more specific, *B4* is a private (non-public) wide area network (WAN), connecting all of Google's data centers for intercommunication across the planet. Since Google is one of the largest internet companies in the world, they are coerced to run a large sized network spanning over multiple continents. A study published in 2010 showed that Google was the top origin autonomous system network (representing the public-facing network) generating the most traffic in 2009 [19] and it is very likely that it's even much higher nowadays. However, Google claims that their internal backbone network grows faster and transports more data than their public-facing

Figure 1: The concept of GGC-requests



Source: Own figure

network (called *B2*), although B2 itself grows at an even faster rate than the Internet [14].

This begs the question what causes those large amounts of traffic and what are the reasons for the intense internal traffic flow. Firstly, large-scale data pushes form the majority of traffic. One single data center cannot be seen as a closed environment, processing only internal data. For reasons of reliability, redundancy and resilience all data centers share the data among them, eliminating a single point of failure in case of data loss or corruption, caused by software or hardware defects, external influences, etc. Therefore, a continuously running synchronizing task ensures the communication and transmission of the data between all data centers over the *B4* network. Secondly, another (less) traffic generating key usage of the WAN is providing data access from remote locations. In most cases, not all data is existing at every single data center. However, for some computations data from distributed sources is requested, which is transmitted over *B4*. Thirdly, user-related data is exchanged between the data centers, increasing the availability. Although this produces much less traffic than the data pushes and remote storage accesses, it is considered as the most priority- and latency-critical task. [14]

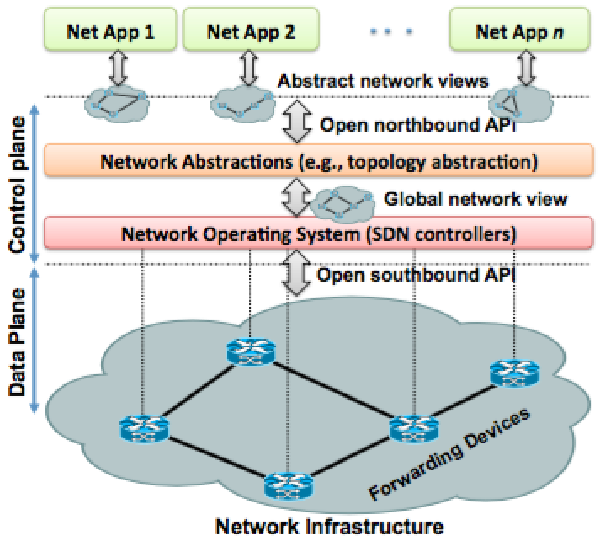
There are multiple reason why Google decided to deploy their own network technologies, turning away from traditional WAN-architectures. The most performant design in terms of priority- and latency-focused transmission is to install dedicated connections between each involved node in the network. That said, this architecture is very expensive to build and it also doesn't scale for a global backbone network. Due to the large distances, including transatlantic or -pacific routes, packet loss between the nodes is harmful regarding the performance, since packets have to be retransmitted over thousands of kilometers having impact on the speed of calculations, results and responses. Another drawback of traditional approaches is the poor efficiency of the network. Assuming that there is no superior global instance controlling and monitoring the traffic flow, it's hardly possible to provide an adequate distribution [13]. Since the network must be capable to manage traffic-peaks, it must be ensured that enough capacity is available. However, the average utilization is 30%-40% [14], resulting in high undersubscription.

B4 is designed in a way that the applications/users are notified about non-preventable errors instead of covering them coupled with retries in the background. But the most significant design principle of *B4* is the switch to **software-defined networking** for reasons of scalability, fault tolerance, global control and cost efficiency [14].

3.2.2 SDN & OpenFlow

Some of the problems of traditional WAN-architectures are already described above, but there are more reasons, why Google decided to redesign their internal backbone: When operating a global network it becomes really hard to manage all networking hardware and gears distributed across the continents. On the one hand this is an issue of distance, because it's necessary to be on site to manually configure the devices. With this in mind, it's also very time-consuming to manage thousands of switches by hand, not to mention

Figure 2: The architecture of SDN



Source: [17]

the error-proneness [2]. Another disadvantage of manual configuration is the diversity of different hardware-models, especially in such large networks. Devices from different manufactures offer different interfaces, commands and functionality, increasing the complexity even more. Software-defined networking helps to reduce the maintaining effort, saving both time and money.

This is accomplished by breaking the vertical integration of the device [17] to gain control over every plane independently, so that the logic can be moved out of the hardware. This approach leads to the architectural design depicted in figure 2.

The SDN-architecture consists of three layers, described bottom-up from device-based to abstraction:

1. **Data Plane:** The data plane is responsible for the transmission of the packets, it's just a simple forwarding device. The data plane does neither manage any rules nor provides further logic. Coupled with the connections between the switches it forms the hardware infrastructure of the network.

2. **Control Plane:** The control plane is located on top of the data plane. It delivers the core functionality of the SDN-driven network, representing the main unit. Unlike the data plane, the control plane does not work on packet-level, but it offers a *system-wide abstraction view* over the whole network.[15] A *Network Operating System (NOS)* running on the control plane instances provides the functionality to manage the complete system. It is responsible to maintain the state of every hardware device, deploy forwarding rules to the switches and communicate with them. [17] The control plane can consist of multiple servers, which can be physically distributed on different locations, providing scalability, availability and responsiveness [4]. But to keep a global overview over the infrastructure it is inevitable to logically centralize the control plane, representing the single instance of control.

3. **Management Plane:** The topmost level in the SDN-architecture is the management plane. It offers the possibility to configure and manage the whole network through an application interacting with the underlying control plane. Thus network administrators don't have to work on the devices directly, but can use a simplified and abstracted level of control in order to update the forwarding rules or monitor the device states. One advantage of this design is the global overview, which is essential in case of failures or problems within the network.

Those three layers provide independent access and management of physical devices, unbound from vendor-specific designs and vertical integration. Due to the separation of planes there must exist a well-defined form of communication between them to control and manage the system. Therefore, the architecture contains programming interfaces, connecting the layers.

- **Northbound API:** The northbound API is responsible for the communication between management- and control plane. For reasons of decoupling and separation, management applications can be developed independently from the underlying structure or design of the control instance. The northbound API may be leveraged as a REST- or SOAP-Interface.

- **Southbound API:** The southbound API works in the same manner, it represents an interface between the control plane and the hardware-based data plane. The southbound API is of vital importance in terms of the SDN infrastructure because it gives access to the devices from a global instance, providing a loose connecting between the separated layers.

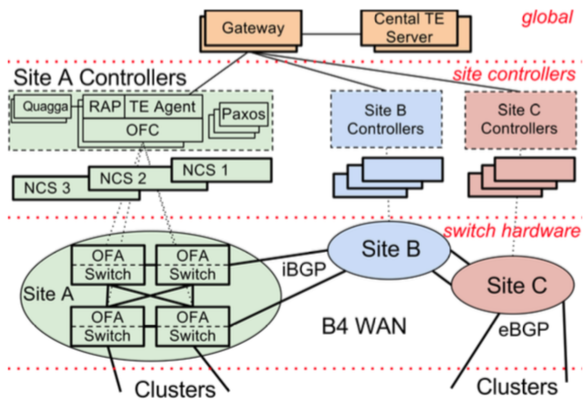
There is a series of communication protocols, which can be applied as southbound API, and *Open-*

Flow is the first and one of the most widely used interfaces [9]. When Google started to rebuild their network as it is today, they were one of the first and largest companies to leverage *OpenFlow* together with SDN. [14] Since the switches in the data plane are just simple forwarding devices without any logic engine, they are reliant upon the commands from the control plane with the help of *OpenFlow*. Each gear holds its own forwarding table, called "*flow table*", since the rules are flow-based (sequencial view of packets between a source and a destination [17]) instead of destination-based. After the communication to the control layer is established, the controller can deploy the rules to the devices. Likewise, the device constantly informs the controller about its current state. When a new packet arrives at the switch, the device will lookup for a matching flow-table entry and forward the packet accordingly. If no matching rule is found, then the default route is applied. In most cases this is the route to the controller, which is responsible for further processing.

In conclusion, a software-defined network offers multiple advantages: Its high agility enables dynamic and fast network changes with very little effort. Administrators don't have to bother with the manual configuration of thousands of devices, they can simply use an application connected to the control plane in order to configure the devices. Abstraction and loose coupling of the individual layers reduces the limitations and constraints of the involved instances while also breaking the vertical integration of the devices. Moreover, there is only one (logically) centralized instance in charge of the network control, in contrast to traditional architectures with no global control at all. Another key thing to remember is the "software-defined" approach, making the network programmable instead of switching wires. With the help of *OpenFlow* as protocol between control and data plane, the network can change its structure within seconds by deploying a new ruleset to the devices. This makes the network scalable (new devices only have to be connected to the controller, which deploys a new rule to the other devices. There is no need for complex wiring anymore) and robust (e.g. deploying additional rules on heavy traffic load (load balancing)) at the same time.

To shift the focus back on B4, the following section describes how Google built their network in a software-defined way and adapted it to their own needs and network characteristics.

Figure 3: The architecture of B4



Source: [14]

Figure 3 depicts the global structure and architecture of their SDN, including additional relevant instances. In the lowest layer, the data center server clusters are hinted, detailed described below (see section 3.3). In this figure there are three data center locations illustrated (from left to right), representing the various data centers across the globe, each having the same infrastructure.

The data plane is represented by the switching hardware decoupled from upper layers, following the SDN-reference. The control plane consists of multiple instances and applications, providing the control functionality. This plane is location-based, which means that there exist individual independent controllers for each site, not directly connected to each other. This design approach is discussed below. The global traffic engineering service is located at the topmost layer, forming a globally centralized unit.

3.2.3 Architecture of B4 in detail

When Google started to build their B4-backbone, they had a lot of things to consider, because building one of the largest networks worldwide is anything but an everyday affair, not to mention they were one of the first and largest companies to make their network software-defined and deploy *OpenFlow*. This threw up a number of problems Google had to solve: Firstly, to operate a software-defined network with *OpenFlow*, it is a prerequisite that the hardware switches are compatible with *OpenFlow* and support the the interface. But since *OpenFlow* was a new protocol and therefore not yet widespread, no vendor produced switches that were able to communicate with the controller. Secondly, it is very expensive to buy thousands of new switches in order to build a new global network and thirdly, B4 has some special characteristics: Although the network covers multiple continents, it only has a few endpoints (the data centers), so the forwarding tables can be kept small and limited. Also, since Google has control over the entire WAN, they can reduce deep-buffering.[14] Considering these aspects, Google decided to go its own way and built their hardware devices from scratch. This approach allowed them to design the switches adjusted to their own requirements and being independent from vendor-specific hardware.

The most notable development while designing the new device is the ***OpenFlow Agent (OFA)*** running directly on the switch. As the name indicates, the OFA is the implementation of the *OpenFlow* protocol adapted for their hardware. It receives commands from the controller and updates the flow table accordingly. It can also redirect packets to the controller, for instance BGP packets are sent to the BGP stack on top [14].

In the site control layer there are multiple ***Network Control Servers (NCS)*** for each site, including the ***OpenFlow Controller (OFC)***. The OFC represents the network controller and can be seen as the central unit of the site-level network. It communicates with all the underlying switches via their OFAs, deploying and updating rules and monitoring their current state to provide an overview over the network. In the case of physical (e.g. relocating switches to another node) or logical (e.g. link failures) network changes, the according OFA sends an update to its OFC to maintain the consistency. The network state is managed in the Network Information Base (NIB), based on the *Onix* platform [16].

Besides the OFC, the NCS also host other service to ensure a reliably running system. This includes *Paxos* [5], a leader election system. At each site, multiple instances of network control servers are distributed over physically separated systems, but only one server is responsible for managing the network. If a failure is detected, the Paxos instances on every server elect a new leader. Another key feature of B4 is the support of traditional shortest-path routing. When Google developed the network, they didn't find themselves in a closed environment without any constraints. As Google was in business since more than a decade before, there were other systems and components which were reliant upon the network, based on traditional protocols. It would have been too cost- and time-critical to redevelop all these systems and make them SDN-compatible, not to mention the extremely high effort to test the systems all together and ensure the reliability of the backbone network further on. That is why Google had no other choice than to support traditional protocols. This is represented by the *Quagga* [20] reference in figure 3. The ***routing application proxy (RAP)*** acts as an interface which helps to integrate *Quagga* and mediate between *OpenFlow* switches and Quagga.

All of the previously described components work on site-level, responsible for the network management at every data center. But the main benefit is derived from a logically centralized (physically distributed across the sites) unit controlling the system as a whole, connected to every site. This is the main function of the global plane in B4's architecture, it represents a global instance which is responsible for the interconnections between the data centers. As depicted in figure 3, the global layer hosts both a gateway an a central traffic engineering (TE) server. The TE service was deployed after the main work on the *OpenFlow* network was finished, abstracted from its implementations and protocols, thus the TE server is layered "on top" of the network as protocol-unaware service. For this reason the gateway exists to enable interaction between the software-defined network and the separated TE server, so that the latter can also control traditional protocol flows.

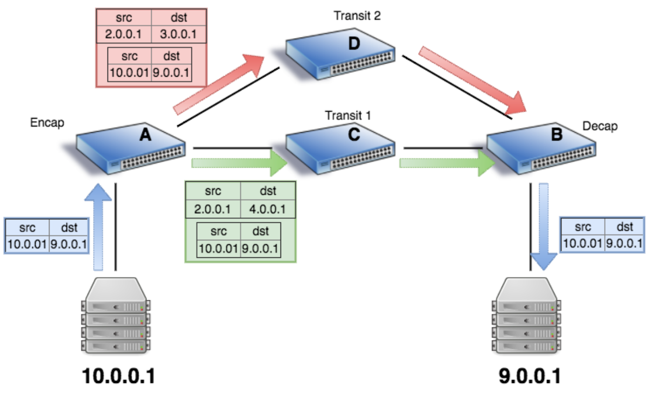
3.2.4 Traffic engineering

Google basically transmits three types of data in their backbone network (see 3.2.1): the most traffic is generated by large-scale data pushes, followed by remote data access and copies of user data. The transmission priority is the other way around with user data having the highest priority. The goal of the TE server is to provide a fair use of bandwidth among all services and applications while taking the previously mentioned considerations into account.

The main problem to solve consists in allocating a fair share of bandwidth to each application. On the one hand it is desirable to provide high utilization of the network, avoiding expensive bandwidth overprovisioning. On the other hand it is important to route the traffic over different various links in order to prevent congestion or bandwidth underprovisioning. This problem is addressed by traffic engineering, which tries to find the best possible routing scheme, taking the services, bandwidth and the current demands into account. This is why the TE server is located on a global layer, it controls the network as a whole.

A so-called *bandwidth-function* determines, how much bandwidth is allocated to which application, based on its relative priority compared to the other involved applications. That way, a *fair-share* is assigned to each application. Since a separate treat-

Figure 4: Multipath WAN forwarding example



Source: Own figure, based on [14]

Table 1: Exemplary Flow Groups, Tunnels and Tunnel Groups

Structure	ID	Value
Tunnel Group	TG-1	{10.0.0.1, 9.0.0.1, QoS}
	...	
Tunnel	T-1	{A,C,B}
	T-2	{A,D,B}
	...	
Flow Group	FG-1	{TG-1, T-1, 0.5}
	FG-2	{TG-1, T-2, 0.5}
	...	

Source: Own table

ment of single applications or packets is too complex, Google defined an aggregation structure (*Flow Group (FG)*) to combine individual application communicating between the same sites, expressed as a tuple with source site, destination site and QoS. Furthermore, they defined two more structures: A *Tunnel* structure defines a path along multiple sites and a *Tunnel Group (TG)* links Flow Groups to the tunnels. Additional weights describe the traffic-ratio of Flow Groups according to each tunnel. [14]. Therefore the principle of *fair-share* can be applied to each Flow Group, allocating fair amount of bandwidth to each FG regarding individual demand and priority. Subsequently the TE server delivers the FGs, Tunnels and TGs to each OpenFlow Controller via the gateway, from where they are deployed on the hardware.

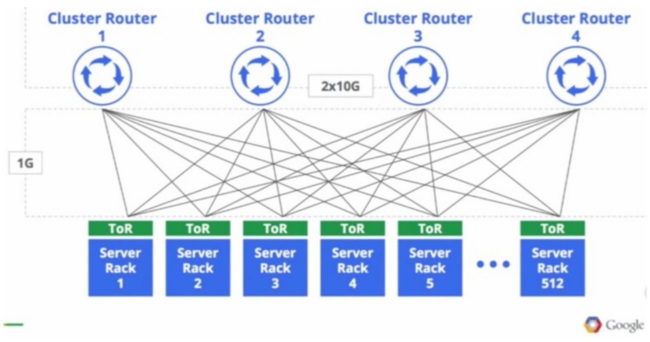
This allows Google to define and dynamically adapt routes with the same source site and the same destination site over different tunnels, resulting in a distribution of traffic across the network. Figure 4 shows the basic idea and functionality behind traffic engineering in B4. It shall be assumed that the source server (IP-address: 10.0.0.1) and the destination server (IP-address: 9.0.0.1) represent two individual sites within Google's backbone. Now two applications running at the source site send data to the destination site at the same time. They both belong to the same Flow Group (provided that they have the same QoS, which is ignored in this example), represented by the tuple {10.0.0.1, 9.0.0.1, QoS}. The first switch chooses for each packet the corresponding TG by looking it up in its internal table (comp. table 1). He finds two entries for the FGs, having different Tunnels assigned. According to the ratio, he sends the first packet based on the first and the second packet based on the second TG entry. This means, the packets are transmitted via a transit switch (C respectively D), so they are assigned a new destination IP (which is actually a tunnel-ID). It is important to mention that it's also necessary to keep the original source- and destination-IP to enable further communication between the endpoints and to ensure the delivery of the packet after decapsulation. For this reason the packets are encapsulated by the first switch, providing a new header with the according IP address of the transit switch derived from the corresponding TG. Therefore, the first packet is encapsulated with the IP address of switch C and the seconds switch with the address of transit switch D. Since each switch keeps the same rules, the transit switches notice that they belong to a tunnel and can than forward the packet to the next node (derived from the tunnel, which belongs to the tunnel-ID). When the packet arrives at the last switch of the tunnel, the switch decapsulates it and can then forward it according to the destination IP.

3.3 Data Center Network Architecture

In this section, the focus is shifted from the data center connecting backbone *B4* to the data centers itself, their architecture and characteristics.

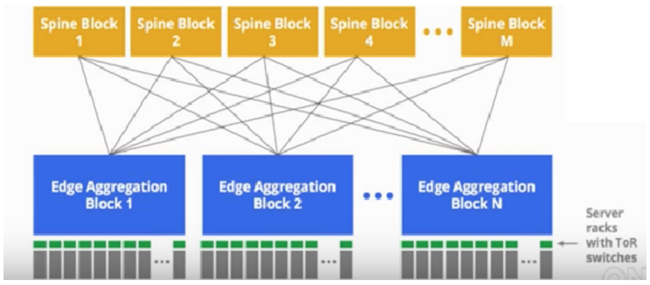
Years ago, the typical consumer-based request for web-resources had a "north-south"-pattern: The request sent by the client was processed by a single webserver and afterwards the response was returned to the client. This approach causes trouble when the request by the user is performance-critical and needs massive computing power, for instance when building the search results. For this reason, another pattern established, called the "west-east"-traffic on top of the "north-south"-traffic: The request is still processed by one single instance, but the background computation is broken down into divided tasks, each handled by another machine. This split of computation leads to lower response times and avoids overutilization of servers. Moreover, it al-

Figure 5: Basic architecture of the four-post cluster architecture



Source: [25]

Figure 6: Basic architecture of Jupiter



Source: [25]

lows the operator to easily scale out, deploying more machines for more power.

Google deployed their first own data center network in 2004, now they use the fifth generation of their network, while they constantly refined and developed their architecture and design. They described three core principles which the data center networks are built upon [24]: Firstly, Google keeps a logically centralized control unit. There are close links to their global backbone B4 (see 3.2), which also follows the SDN approach. The switching logic is separated from the hardware devices within the data center to the control plane (see fig. 3). This allows Google's system to dynamically set and manipulate the forwarding rules. Secondly, the company was not contented with their reliance and dependence on third-party vendors producing switching hardware. The need for a custom design and architecture tailored to their own requirements led Google to the decision to build their own switching components. On the one hand the engineers can ignore the support for unnecessary and thus unused protocols, while they also can include additional, custom functionality, for instance the implementation of the OpenFlow-Agent (see 3.2.3), on the other hand. Thirdly, and this is the most conspicuous approach, Google designed their (later generation) networks to adopt a Clos topology. The Clos topology provides scalability and reliability by adding additional stages to the network.

The first architecture deployed by Google in 2004 was the so-called "four-post" cluster architecture [24] depicted in figure 5. This design can be seen as a leaf-spine topology, where the top-tier switches (cluster router) represent the spine of the network and the underlaying Top of Rack (ToR) switches represent the leaves. The leaves are not connected to each other as well as the core routers are not interconnected. The main problem with this approach was primarily the weak hardware. The transmission capacity was limited to a bandwidth of 100 Mbps per host, resulting in packet drops [24] (at that time they did not yet build and leverage their own switches). One solution could have been to reduce the cluster size and therefore increase the bandwidth of each server, but this involves very high costs. Another approach could have been to scale out and add more clusters the spines, but this was not possible, too. Each cluster router had 512 1G ports, each connected to one of the ToR switches, containing up to 40 servers. This means that the total maximum amount of servers was reached with the number of 20.480, any additional server would have required additional ports for each core router.

Instead of investing in an inflexible, low-bandwidth, unscalable data center network, Google decided to reconsider their needs and develop their architecture further. At that time, Google began to identify the three key characteristics described above and build their new architectures upon the centralized control protocols, the Clos topologies and the merchant silicon. As a consequence, they designed multiple generations of data center networks over the years, the most recent one called "Jupiter" was first deployed in 2012.

As can be seen in figure 6, the most conspicuous change between the first generation and following architectures is the additional stage between spine and leaves. This middle-layer is called the *aggregation stage*, since it combines multiple clusters into separated blocks. Each aggregation block is connected to each spine block, solving the main problem of scalability. The spine block offers 128 40G ports to the underlying aggregation block. However, for reasons of redundancy the aggregation block consists only of 64 individual blocks. Each of those aggregation blocks contains eight so-called *"Middle*

Blocks" (MB) with a 40G bandwidth. The lower ToR-switches are connected to the Middle Blocks via 16 10G ports and to the underlying servers via 48 10G ports.

This redesigned network architecture with the powerful hardware has great impact on the overall performance of the company's data center networks. While the first generation four-post cluster architecture had a total bisectional bandwidth [1] of 2 Tbps, Google managed to increase the bandwidth of *Jupiter* by a factor of 650 to 1.3 Pbps. [24]

4 Conclusion

This paper provides an overview over the most important characteristics and instances within Google's global network. Google is one of the largest tech-companies worldwide, therefore they are reliant on a globally available, performant and cost-effective network connecting the internal and external nodes. With the *Google Global Cache* servers deployed at ISPs connected with the numerous data centers, which in turn are connected via a global private WAN, Google designed and built a system of coherent instances. Those networks are well integrated into the surrounding ecosystem, depending on each other and forming high synergy. Although Google created powerful and efficient networks that way, they did not leverage completely new technologies and did certainly not revolutionize networking: Clos topologies in telecommunications are more than 60 years old [7] and software-defined networking also had its first appearance more than a decade ago [8]. Nevertheless, Google was one of the first and largest companies to leverage SDN and *OpenFlow* and they also developed their own *OpenFlow Agent* in combination with their own hardware. All things considered, Google's networks represent a (role-)model for many other large companies (*Facebook* started to build their own gear nine year after Google did [23]), while its development in networking structures, architectures and concepts is more a evolution than a revolution.

References

[1] Alphabet. *Umsatz von Alphabet weltweit vom 1. Quartal 2008 bis zum 1. Quartal 2016 (in Millionen US-Dollar).* In: Statista - Das Statistik-Portal. 2016. URL: http://de.statista.com/statistik/daten/studie/154635/umfrage/umsatz-von-google-quartalszahlen/ (visited on 07/26/2016).

[2] Theophilus Benson, Aditya Akella, and David Maltz. "Unraveling the Complexity of Network Management". In: *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation.* Boston, Massachusetts: USENIX Association, 2009, pp. 335–348. URL: http://dl.acm.org/citation.cfm?id=1558977.1559000.

[3] Eric Brewer et al. *Disks for Data Centers.* Tech. rep. Google, 2016. URL: http://research.google.com/pubs/pub44830.html.

[4] M. Canini et al. "A distributed and robust SDN control plane for transactional network updates". In: *2015 IEEE Conference on Computer Communications (INFOCOM).* Apr. 2015, pp. 190–198.

[5] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. "Paxos Made Live: An Engineering Perspective". In: *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing.* PODC '07. ACM, 2007, pp. 398–407. URL: http://doi.acm.org/10.1145/1281100.1281103.

[6] Cisco. *The Zettabyte Era—Trends and Analysis.* June 2, 2016. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html (visited on 07/21/2016).

[7] C. Clos. "A Study of Non-blocking switching networks". In: *Bell Syst. Tech. J.* 32 (1953), pp. 406–424. URL: https://ia801602.us.archive.org/11/items/bstj32-2-406/bstj32-2-406.pdf (visited on 07/29/2016).

[8] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The Road to SDN: An Intellectual History of Programmable Networks". In: *SIGCOMM Comput. Commun. Rev.* 44.2 (Apr. 2014), pp. 87–98. URL: http://doi.acm.org/10.1145/2602204.2602219 (visited on 07/29/2016).

[9] Open Networking Foundation. *OpenFlow.* 2016. URL: https://www.opennetworking.org/sdn-resources/openflow (visited on 07/17/2016).

[1] Bisection bandwidth describes the bandwidth at a given point within the network, at which the network is split in two parts, while both parts have the same size

[10] Google. *Edge Nodes (GGC).* n.d. URL: https://peering.google.com/#/options/google-global-cache (visited on 07/15/2016).

[11] Google. *Peering @ Google - Our Infrastructure.* n.d. URL: https://peering.google.com/#/infrastructure (visited on 07/13/2016).

[12] Google. *Standorte von Rechenzentren.* n.d. URL: https://www.google.com/about/datacenters/inside/locations/index.html (visited on 07/13/2016).

[13] Chi-Yao Hong et al. "Achieving High Utilization with Software-driven WAN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM.* SIGCOMM '13. Hong Kong, China: ACM, 2013, pp. 15–26. URL: http://doi.acm.org/10.1145/2486001.2486012.

[14] Sushant Jain et al. "B4: Experience with a Globally-deployed Software Defined WAN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM.* SIGCOMM '13. ACM, 2013, pp. 3–14. URL: http://doi.acm.org/10.1145/2486001.2486019.

[15] Manar Jammal et al. "Software-Defined Networking: State of the Art and Research Challenges". In: *CoRR* abs/1406.0124 (2014). URL: http://arxiv.org/abs/1406.0124.

[16] Teemu Koponen et al. "Onix: A Distributed Control Platform for Large-scale Production Networks". In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation.* USENIX Association, 2010, pp. 351–364. URL: http://dl.acm.org/citation.cfm?id=1924943.1924968.

[17] D. Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.

[18] Craig Labovitz. *Google Sets New Internet Record.* July 22, 2013. URL: https://resources.arbornetworks.com/h/i/29325532-comment-on-how-big-is-google-by-deepfield-blog-google-sets-new-internet-record-bespacific-%C3%83%C2%A2-bespacific (visited on 07/18/2016).

[19] Craig Labovitz et al. "Internet Inter-domain Traffic". In: *Proceedings of the ACM SIGCOMM 2010 Conference.* SIGCOMM '10. ACM, 2010, pp. 75–86. URL: http://doi.acm.org/10.1145/1851182.1851194.

[20] Quagga. *Quagga Routing Suite.* 2015. URL: http://www.nongnu.org/quagga/ (visited on 07/18/2016).

[21] Sandvine. *Global Internet Phenomena - Asia-Pacific & Europe.* 2015. URL: https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-report-apac-and-europe.pdf (visited on 06/29/2016).

[22] Sandvine. *Global Internet Phenomena - Latin America & North America.* 2015. URL: https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-report-latin-america-and-north-america.pdf (visited on 06/29/2016).

[23] Adam Simpkins. *Facebook Open Switching System ("FBOSS") and Wedge in the open.* 2015. URL: https://code.facebook.com/posts/843620439027582/ (visited on 07/26/2016).

[24] Arjun Singh et al. "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network". In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 183–197.

[25] Amin Vahdat. *ONS 2015: Wednesday Keynote - Amin Vahdat.* 2015. URL: https://www.youtube.com/watch?v=FaAZAII2x0w (visited on 07/25/2016).