An Analysis of Load Balancing Algorithms in Web Applications

J. Gledhill
Department of Computer Science
University of Sheffield
Sheffield, UK

Abstract: Every day, websites like Google, Instagram and many other household names receive millions or even billions of requests from their users. This level of demand is far too much for a single machine to handle, so it must be distributed across thousands of servers all connected by a load balancer. This report investigates the use of load balancing in industry and the design considerations required to implement it into a web application. It presents four industry-standard algorithms, as well as analysing which one is ideal for certain levels of traffic. Round robin is the simplest of these but provides no fault tolerance and quickly becomes obsolete. IP Hash can be used for geolocation but may overload servers closest to large groups of users. Least Connections/Response Time is very versatile, and provides decent fault tolerance, but is not suitable at enterprise scales. Resource-based provides the best fault tolerance and can run for a long time without human intervention but is by far the most complex and demands a hefty investment. It was found that higher levels of traffic require more expensive and complex load balancing solutions, while smaller levels may not even require load balancing at all. Thus, fast growing companies must closely monitor their capacity, upgrade their hardware and make investments to future-proof their infrastructure to prevent a catastrophic failure.

Keywords: load balancing, parallel processing, distributed computing, overload prevention, fault tolerance, web applications

1. INTRODUCTION

For many popular and high-traffic web apps, such as Google, Microsoft, Amazon, and many others, it is often impractical to respond to every single visitor through a single machine or server [1]. This is due to an upper limit on the capacity of a single server; the hardware can only handle so much before no more users can be served [1]. So instead, large tech companies run their systems across a huge number of servers, sharing enormous amounts of data; Google operates 31 datacentres (server warehouses) worldwide, Microsoft has over 4 million servers in operation, and Facebook processes on average 700 petabytes of data from millions of users every day [2-4].

This scale of data and processing needs creates a big problem for engineers: how can all these servers be connected together? Load balancing is one of the solutions to this problem; it works by routing each visitor to one of many possible servers that all act as the same web app, this is known as a "(server) pool" or "(server) farm" [1, 5-6]. The technicalities in how this is achieved differ between web apps, some may even use proprietary load balancers, so this report will investigate the most common and well-researched solutions. The overall goal of load balancing is to ensure a web app is almost always available to most of its users [1, 5-6]. There may also be secondary goals including efficient resource usage, network security, reduced operational cost, improved sustainability, and easier maintenance [1, 5-6].

2. TECHNICAL CONTENT

There are many different load balancing algorithms that are used in the industry, and some larger corporations may even develop their own proprietary algorithms. That said, there are several algorithms that are used commonly across industry to implement effective, 'tried-and-tested' load balancing, without the difficulty of designing and developing a proprietary algorithm. In general, a load balancing algorithm can be said to be either dynamic or static; static load balancing algorithms are independent of the state of the server farm and follow a fixed set of rules, they are typically simpler than dynamic algorithms [6-7]. Hence, dynamic load balancing algorithms are the inverse; they adjust their output according to current network demand and server health, making them less predictable and more complicated [6-7].

2.1 Server Weighting

Many of these algorithms can utilise weighting to direct proportionally more traffic to servers with a higher weight (see Figure 1 for a visual demonstration) [5-7]. This allows administrators to route more traffic to servers that can handle a higher number of users compared to other servers in the pool [5-7]. Weightings can also be decided by other factors,

such as priority or location [6]. It could be argued that weightings can be used, on otherwise static algorithms, to achieve a 'pseudo-dynamic' algorithm - providing the simplicity of a static algorithm, with many of the benefits of dynamic algorithms. Administrators can also decide not to implement weighting into the algorithm at all, be it for simplicity or because weighting is unlikely to provide much benefit (e.g. if all servers have roughly the same processing capacity).

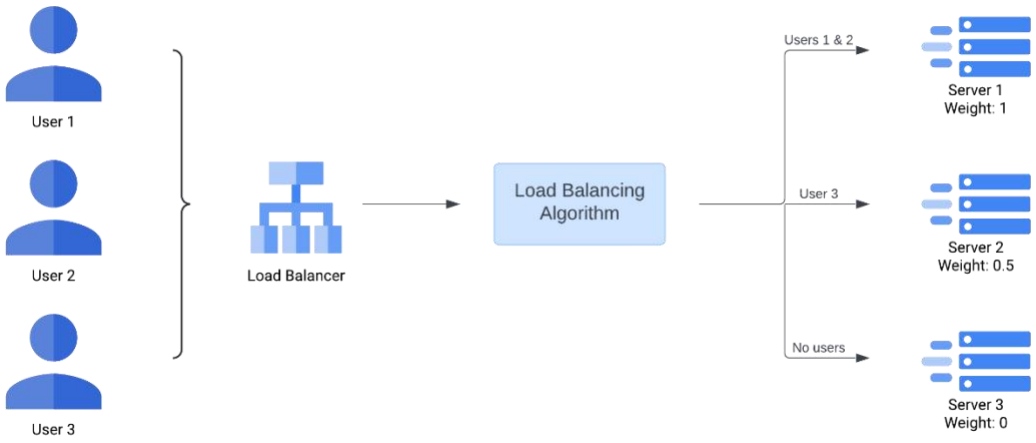

Figure 1 - Demonstration of weighting distributing proportionally more users to servers with higher weights

2.2 Round Robin Algorithm

This is a static algorithm and is perhaps the simplest algorithm in use today. It works by distributing traffic evenly across a list of available servers from a server pool, typically by progressing down the list with each new user until it starts again once the list is exhausted – hence the name 'Round Robin' [1, 5-7]. This algorithm is often weighted to allow more powerful servers to receive more requests [5-7].

In web apps, this algorithm is frequently implemented by instructing the Domain Name System (DNS) to rotate through the list [5, 7]. The DNS is the system used by almost all web apps to route a user's request from a domain (e.g. google.com), to the IP address of the server that will fulfil the request [8]. It does this by returning what are known as A ("address") records that map a given domain name (for example, google.com) to a single IP address [8]. The server(s) that manage and provide these DNS records are typically referred to as Authoritative Name Servers (ANS) [8]. In the case of the Round Robin algorithm, the ANS will rotate through a list of A records that all map to the same domain name [8]. Since an A record can only contain a single server IP address, the ANS must change the A record every time a new request is received. Because this algorithm makes use of the DNS, web apps often make use of third-party DNS providers (such as Cloudflare or other domain registrars) rather than running their own load balancers [9].

Alternatively, web apps may also use AAAA records instead of A records if they're using IPv6 addresses instead of IPv4 addresses [8]. IPv4 addresses make up the majority of IP addresses in use today, and have done so for several decades [10]. These addresses take the form of 'X.X.X.X', where each X is an 8-bit integer (for a total of 32 bits), meaning IPv4 has approx. 4.29 billion addresses [10]. In comparison, IPv6 has a total of 128 bits per address, giving a total address space of approx. 3.4×10^{38} addresses, solving the problem of potentially running out of IP addresses (as with IPv4) for a very long time [10]. Eventually, as the number of devices in use on the internet continue to grow, all IPv4 addresses will finally be exhausted, and the world must switch to using IPv6 addresses instead [10]. Since the algorithm only needs to rotate through a pre-validated list of IP addresses (provided by a network admin), a web app's use of IPv4 or IPv6 has no impact on how this algorithm functions.

Figure 2 demonstrates this algorithm in action and shows the sequential order in which the ANS rotates through the server list.

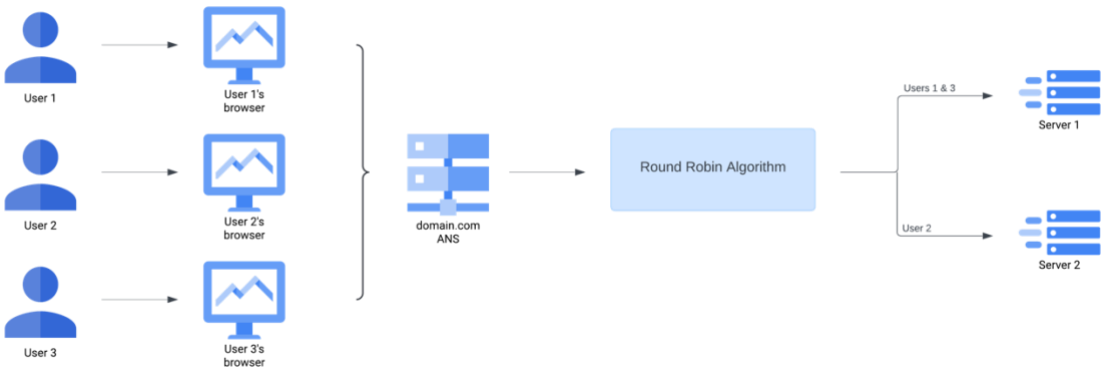

Figure 2 - Demonstration of the Round Robin algorithm load balancing 3 users across 2 servers

2.3 IP Hash Algorithm

This is another type of static algorithm that works by using a mathematical computation (called a hashing algorithm) to produce a hash key that maps the incoming user's IP address to one of the servers in the server pool [1, 5-7]. Because of this, users with static IP addresses will always arrive at the same server(s) – regardless of availability or performance. Most Internet Service Providers (ISPs) periodically change their customers' IP addresses (hence it is dynamic), via a Dynamic Host Configuration Protocol (DHCP) server, as a means of protecting the customer's identity [11]. This is because static IP addresses can be used to identify a user's location [11]. However, some ISP customers (in particular, those running servers or datacentres), request static IP addresses [11]. This is because a dynamic IP address would cause downtime if the algorithm is not designed to constantly monitor the server's IP address, and update accordingly.

IP geolocation (as shown in Figure 3) is an example of a hashing algorithm that may be used in this load balancing algorithm, where all the IP addresses within a region are matched to a hash key [12]. It uses databases mapping IP ranges to a certain country or region, then produces a hash key that corresponds with the server(s) geographically closest to the user [12]. For a large, multinational web app like Google, using IP geolocation with this algorithm could allow them to distribute traffic across their 31 datacentres (located anywhere from South America to Western Europe to Southeast Asia) according to which is closest to the user [2].

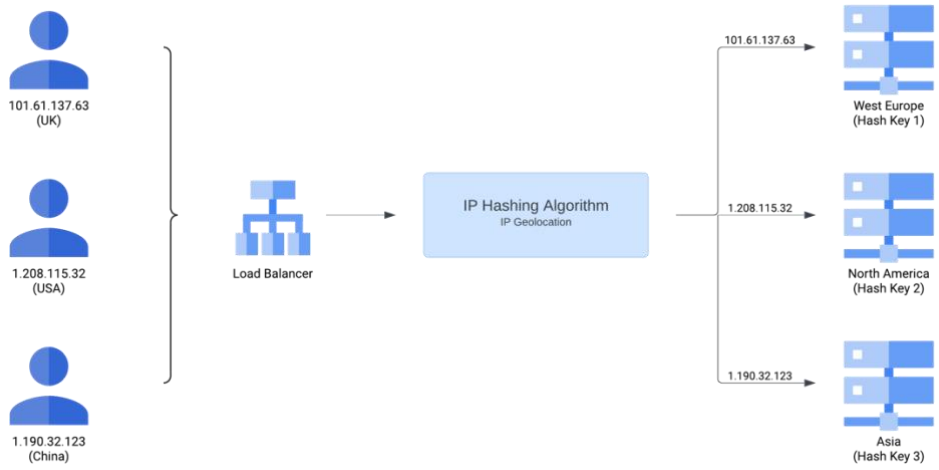

Figure 3 - IP Hash algorithm using internet geolocation to route users to the geographically closest servers

2.4 Least Connections & Least Response Time Algorithms

As the name implies, this algorithm prioritises the server(s) with the fewest open connections with the aim to distribute traffic evenly [5-7]. This makes it similar in outcome to the Round Robin algorithm but differs in the execution; Round Robin is static, Least Connections is dynamic. Like Round Robin, this algorithm also assumes that all servers in the pool have roughly equal capacity, and that all open connections require equal processing power [6]. In situations where some servers have greater capacity than others, weighting can be used to direct more traffic to more powerful servers [6-7]. Figure 4 demonstrates this, with all connections being sent to the server with the highest weight (regardless of connections), before lower weight servers begin to receive connections.

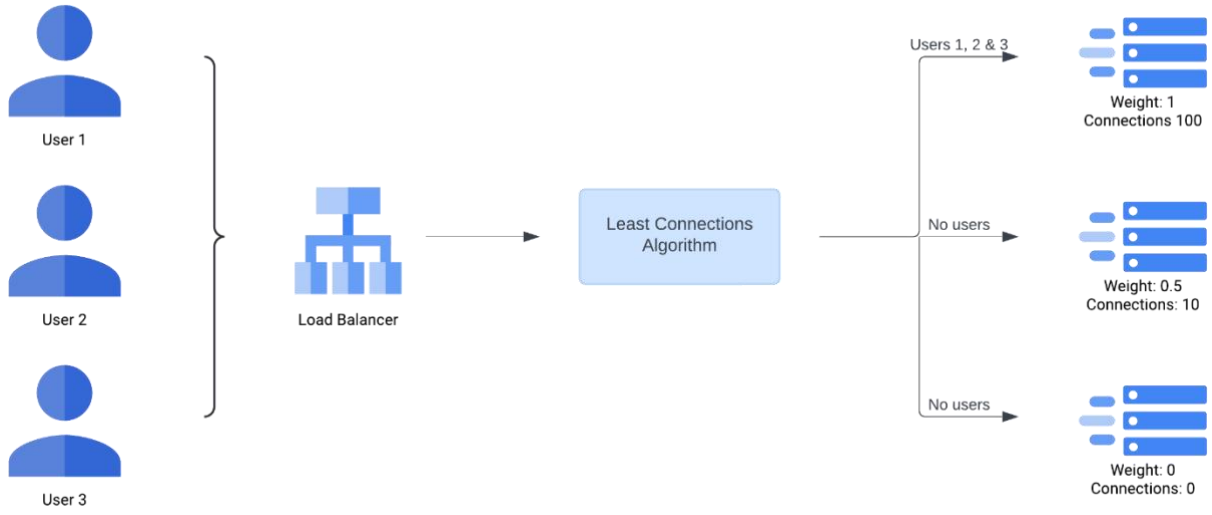

Figure 4 - Weighted Least Connections algorithm directing all users to the server with the highest weight, despite having the highest number of connections

Least Response Time is a variation of Least Connections that also factors in a server's average response time when determining where to route users [5-7]. This allows the algorithm to direct traffic to the servers that will respond fastest, while also detecting when servers are close to being overloaded with too many active connections [6-7]. This algorithm can be weighted in the same way that Least Connections can [5-7].

2.5 Resource-based Algorithm

The Resource-based algorithm collects information about resource usage of each server in the pool and uses the information to route traffic appropriately [6-7]. Exactly which server the traffic is routed to depends on the specifics of the algorithm, it might route to the server with the most available resources, or it might route to any server that is using less than a threshold utilisation. Servers which have too many available resources (and hence, not processing enough requests) are said to be underloaded, while servers which have too few available resources and cannot handle any more requests are said to be overloaded. Being between underloaded and overloaded is optimal, because it makes efficient use of the resources without damaging the user experience.

Each server has a piece of software installed on it that communicates the server's CPU, memory and other resource utilisation data to the load balancer, this is called a Software Agent [6-7]. Software Agents could also be designed to gather data from other algorithms for the load balancer to use, such as average response time or active connections. When engineered properly, this algorithm can provide fault tolerance and overload prevention to the system without any additional work required. This is because the Software Agents can provide the load balancer with all the data required to detect overloaded or faulty servers and take reasonable action without human interference. Figure 5 demonstrates this algorithm prioritising underloaded servers and shows the Software Agents feeding information to the algorithm.

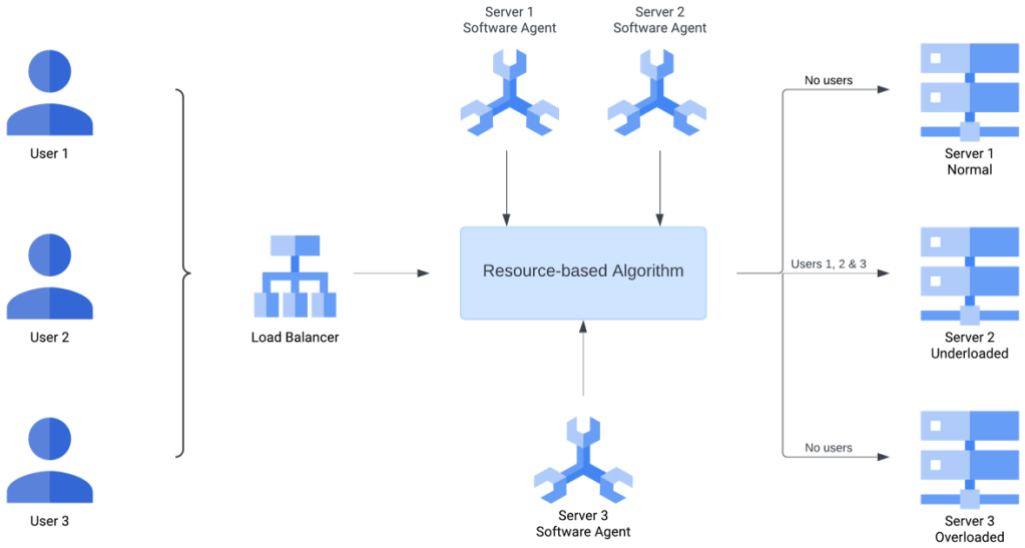

Figure 5 - Resource-based algorithm directing all traffic to the server with the most available resources

3. ANALYSIS AND KEY ISSUES

This section shall address the options and considerations presented previously and discuss them within the context of three scales of web apps. These scales aim to address the differences in limitations and resources between different levels of web app traffic. Namely, factors like revenue, access to highly skilled engineers, and user demand are taken to account.

3.1 Enterprise Scale Web Applications

These web apps process billions of requests daily and are at the apex of web apps. They are used by millions of users worldwide and thus are expected to be available 24/7/365 with negligible downtime. These web apps are owned by multinational corporations (e.g. Microsoft, Amazon, Netflix), with hundreds of billions of dollars in annual revenue and access to highly educated engineers.

For these web apps, constant uptime of utmost importance. Multiple datacentres will be required to handle the vast number of users and thus the system needs to be highly redundant and self-sufficient. Web apps of this scale should use a nested load balancing system, where traffic is first distributed to a datacentre before being distributed to individual servers. This central load balancer could use the IP Hash algorithm to distribute traffic across datacentres according to their proximity to the user. The Resource-based algorithm is essential for providing the level of fault tolerance and overload prevention necessary to prevent disruption to end-users, as well as coping with spikes in activity or DDoS attacks (where attacks attempt to flood the servers with requests to cause an overload). Multiple hardware load balancers could be installed to provide the throughput necessary, or many virtual load balancers could be used instead for added flexibility and automation.

3.2 Large Scale Web Applications

Large scale web apps do not receive as many requests as enterprise scale web apps but are still expected to be available most of the time with very short periods of downtime. Hundreds of thousands of users access these web apps daily and generate millions of requests. These web apps account for a larger proportion of the internet than enterprise scale web apps, but not as much as small & medium scale web apps. Examples of web apps that would fall in this category include the UK government website, the University of Sheffield's student hub, and the National Rail website. The companies that own these web apps will receive millions or billions in yearly revenue, with access to highly skilled engineers.

Due to the traffic these web apps can expect to receive, it is likely that many of them will require at least one datacentre to operate. They may be at risk of activity spikes, particularly if they have a local userbase, so would likely benefit from designing their system to prevent overloading and tolerate faults. These web apps should run virtual load balancers on one of their physical servers in the datacentre to allow for growth without having to repeatedly purchase new hardware. The Least Connections & Least Response Time algorithms will likely provide a satisfactory experience for users, without the added complexity of engineering and installing Software Agents, as in the Resource-based algorithm.

However, these web apps may grow to the extent that they become enterprise scale web apps. If this is not planned for in advance, this could lead to large spikes in activity and demand that could grow too fast for engineers to keep up with. This would likely lead to a mass overload of all the web app's servers, causing downtime and leaving engineers panicking to install new hardware quickly so that the web app can be restarted. As a result, it is important that engineers closely monitor growth and always keep a large buffer between current demand and capacity. In many cases, this could be as simple as buying more servers than are needed. However, longer term future-proofing could involve using the Resource-based algorithm, which can scale much better than others and will provide extra fault tolerance to reduce engineer intervention. This will require an investment from the company, and not being able to make this investment could cause the web app to fail and become unprofitable.

3.3 Small & Medium Scale Web Applications

These web apps form the majority of web apps on the internet and will have no more than a few thousand daily users. They are unlikely to need to be available 24 hours a day and typically include personal portfolio webpages, small business or charity websites, local authority websites, amongst many others. Most of the companies or individuals operating these will have no more than a few million in revenue and do not require highly skilled engineers to operate them properly.

In the case of these web apps, it is unlikely that load balancing is necessary at all. This is because the level of traffic is unlikely to exceed the capacity of a single server, and thus there is no need to distribute traffic across multiple servers. However, in cases where it is justified, the Round Robin algorithm is likely to be the most cost-effective solution due to the simplicity of the algorithm and the lack of a dedicated hardware/software appliance. Additionally, the algorithm can be weighted should any servers have significantly more capacity than others. While this does not provide any active overload prevention, the passive overload prevention from the algorithm should prove satisfactory in dealing with most traffic spikes. Alternatively, a cloud load balancing platform may be preferred due to the ease-of-use and the smaller, subscription-model costs are likely to be more manageable for these web apps.

4. CONCLUSIONS

Overall, it is clear that the design of a load balancing system closely relates to the amount traffic that the web app attracts, and the amount of revenue available to the company. More popular web apps require more complex solutions and architectures to cope with the high number of users, while smaller web apps may not even need to be load balanced at all. Engineers should consider the expected activity and forecasted growth of their web app when considering which algorithm to implement, the level of overload prevention and fault tolerance required, and whether to purchase dedicated appliances/software licenses or opt for a cloud load balancing platform instead. For companies experiencing fast growth of their web apps, future-proofing their infrastructure should be a priority, this is to prevent sudden failure and catastrophic overload that could jeopardise the company's users and profits. This can be done by investing in more servers, creating a large buffer between current user demand and full capacity, hiring skilled engineers to develop more advanced algorithms like Resource-based to ensure the load balancer can keep up with capacity and avoid a bottleneck.

5. REFERENCES[1]

[1] F5, Inc., "What is a load balancer." F5. https://www.f5.com/glossary/load-balancer (Accessed: Mar. 19, 2024).

[2] Google, "Discover our data center locations." Google Data Centers. https://www.google.com/about/datacenters/locations (Accessed: June. 9, 2024).

[3] J. Roach, "Microsoft's virtual datacenter grounds 'the cloud' in reality," *Microsoft*, Apr. 2021. [Online]. Available: https://news.microsoft.com/source/features/innovation/microsofts-virtual-datacenter-grounds-the-cloud-in-reality
[4] J. Flinn and A. Aggarwal, " [4]," *Engineering at Meta*, July. 2022. [Online]. Available: https://engineering.fb.com/2022/07/14/data-infrastructure/owl-distributing-content-at-meta-scale/

[5] IBM, "What is load balancing." IBM. https://www.ibm.com/topics/load-balancing (Accessed: Mar. 17, 2024).

[6] Amazon Web Services, Inc., "What is load balancing." AWS. https://aws.amazon.com/what-is/load-balancing/ (Accessed: Mar. 19, 2024).

[1] IEEE standard

[7] Cloudflare, Inc., "Types of load balancing algorithms." Cloudflare. https://www.cloudflare.com/en-gb/learning/performance/types-of-load-balancing-algorithms/ (Accessed: Mar. 19, 2024).

[8] Cloudflare, Inc., "DNS A record." Cloudflare. https://www.cloudflare.com/en-gb/learning/dns/dns-records/dns-a-record/ (Accessed: May. 22, 2024).

[9] Cloudflare, Inc., "What is round-robin DNS." Cloudflare. https://www.cloudflare.com/en-gb/learning/dns/glossary/round-robin-dns/ (Accessed: May. 27, 2024).

[10] A. N. A. Ali, "Comparison study between IPv4 & IPv6," *International Journal of Computer Science Issues*, vol. 9, no. 3, pp. 314-317, May. 2012. [Online]. Available: https://www.proquest.com/scholarly-journals/comparison-study-between-ipv4-ipv6/docview/1030094256/se-2

[11] F. Lah, "Are IP addresses 'personally identifiable information'," *Journal of Law and Policy for the Information Society*, vol. 4, no. 3, pp. 681-707, 2008. [Online]. Available: https://kb.osu.edu/server/api/core/bitstreams/e9b18766-1e82-5c8c-a21a-6fa7a48964d5/content

[12] J. Saxon and N. Feamster, "GPS-Based Geolocation of Consumer IP Addresses," in *International Conference on Passive and Active Network Measurement*, 2022, pp. 122-151. [Online]. doi: 10.1007/978-3-030-98785-5_6.