Decentralized CDN for Video Streaming

Matias Correia
Mestrado Integrado em Segurança Informática
Departamento de Ciências de Computadores
2022

Orientador
Prof. Dr. Rolando Martins, Faculdade de Ciências

Coorientador
Prof. Dr. Luís Antunes, Faculdade de Ciências

Todas as correções determinadas

pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/________

DECLARAÇÃO DE HONRA

Eu, Matias de São José Rosa Ramalho Frazão Correia, inscrito no Mestrado em Segurança Informática da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Assinatura do Autor

Data 21/09/2022

UNIVERSIDADE DO PORTO

MASTERS THESIS

Decentralized CDN for Video Streaming

Author:
Matias CORREIA

Supervisor:
Rolando MARTINS

A thesis submitted in fulfilment of the requirements
for the degree of MSc. Cyber Security

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Ciências de Computadores

January 23, 2023

" I am and always will be the optimist, the hoper of far-flung hopes and the dreamer of improbable dreams "

Matt Smith as *The Doctor*, written by Matthew Graham

Acknowledgements

I want to take a moment to thank all the people who helped me through this journey.

First, I wish to thank my supervisor, Prof. Dr. Rolando Martins. Without his guidance it would have been impossible.

A special thanks to Steven Allen, one of the lead developers in IPFS, who always answered my emails helping me understand the core logic behind IPFS.

To my friends and girlfriend, which supported me through the dark times. We can now be together without you having to ear me complaining about all the tests gone wrong.

Finally to my family for their unconditional support, patience and understanding. Without you believing in me, I never would have made it. This accomplishment is also a bit yours.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto

Departamento de Ciências de Computadores

MSc. Cyber Security

Decentralized CDN for Video Streaming

by Matias CORREIA

The increase in video streaming consumption, representing a large portion of internet global traffic, has risen the costs for this service providers Content Delivery Network (CDN) infrastructures. This dissertation explores the option of using Peer-to-Peer (P2P) to share cached video files and the future integration of this group with standard CDN infrastructures, diminishing the costs associated with the bandwidth used by the servers.

We use InterPlanetary File System (IPFS)[1] which is a P2P hypermedia protocol that aggregates different protocols creating a decentralized file sharing system. We applied and tested some changes to IPFS in order to potentially improve its performance in video file sharing since streaming is a delay sensitive task.

To test the changes made to IPFS we measured 3 key metrics: the delay between the requested and the sending of a block, the delay between the sending and the reception of a block and the number of duplicate blocks in each test.

As a result, the margins between tests were very slim. The reason for this could be because of the random factor used, by default Bitswap and by our implementation, when selecting the peers to provide blocks.

To have a better grasp of the impact of the changes made to IPFS more tests on each scenario are necessary for a trend to be observable. Also tests that would be closer to real world scenarios with more nodes could also lead to better conclusions. Nonetheless IPFS is a viable option to form a P2P group for video streaming and sharing.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Ciências de Computadores

Mestrado em Segurança Informática

Optimização de custo em streaming de vídeo

por Matias CORREIA

O aumento no consumo de *streaming* de vídeo, que representa uma larga porção do tráfego global de internet, tem expandido os custos para as infraestruturas das *Content Delivery Network* (CDN) destes provedores de serviços. A presente dissertação explora a opção de utilizar *Peer-to-Peer* (P2P) para partilhar os ficherios de vídeo guardados em cache e a futura integração deste grupo com a infraestrutura de uma CDN standard diminuindo assim os custos associados com a largura de banda usada pelos servidores.

Usamos o InterPlanetary File System (IPFS)[1] que é um protocolo P2P de *hypermedia* que agrega diferentes protocolos para criar um sistema de ficheiros descentralizado. Foram feitas e testadas mudanças ao IPFS para potencialmente melhorar a sua performance no que toca à partilha de ficheiros de vídeo, uma vez que o *streaming* é sensível à latência.

Para testar as mudanças feitas ao IPFS medimos três métricas-chave: a latência entre quando um block é pedido e enviado, a latência entre quando um block é enviado e recebido e o número de blocos duplicados em cada teste.

Concluímos que as margens entre cada teste foram pequenas. Isto pode dever-se ao fator aleatório usado, pelo Bitswap default e pela nossa implementação, na escolha do próximo *peer* para fornecer os blocos.

Para perceber melhor o impacto das mudanças feitas ao IPFS seriam precisos mais testes em cada cenário para criar uma tendência observável. Além disso, testes com mais nós na rede, num cenário mais próximo de uma utilização real do protocolo, poderia levar a que fossem tiradas melhores conclusões. Contudo, o IPFS é uma opção viável para formar um grupo P2P para partilha e *streaming* de vídeo.

Contents

List of Figures

Chapter 1

Introduction

Cloud computing, an extremely popular solution based on centralization of computation and storage, has recently been threatened by the large growth of devices with internet access. With the development of Internet of Things (IoT), the introduction of new gadgets to the market and the continuous rise of smartphones, cloud services infrastructure has been pushed to the limit in order to provide the Quality of Service (QoS) required to real time applications.

Nowadays, video consumption is at an all time high. The growth and development of mobile devices alongside with apps like Youtube and Netflix, which have seen a big influx of new users in the past years, have contributed to the exponential growth in the amount of video streaming internet traffic. For example, in February 2020, Youtube was estimated to have 500h of video uploaded every minute [2] and Cisco estimates that in 2022 82% of internet traffic will be video streaming [3].

With currently large amounts of traffic being generated on a wide range of devices, connections to cloud data centers need to be larger and faster and thus more expensive in order to provide adequate latency to maintain QoS. A number of technologies, such as Cloudlet [4],[5], Mobile Edge Computing (MEC) [6], Fog computing [7] and Mobile edge-clouds [8], have emerged to bring the cloud closer to the end user, aiming to provide a reduced delay between requests and also to reduce the burden on centralized cloud servers, distributing the traffic by placing more servers on the edge of the network. The adoption of these technologies comes with the rise of more demanding real time applications requiring a better latency to provide good Quality of Experience (QoE) to their users.

With an even more distributed approach in mind, Peer-to-Peer (P2P) provides a way of distributing furthermore this traffic amongst the service users reducing the backhaul burden on cloud servers, reducing latency, providing a cheaper service and possibly even with less delays.

Cyber security is a big concern in modern world technology and users want to be safe with their data. As cloud services protect their user data, decentralized solutions also have to do too. A P2P technology have to ensure that users do not have access to their peers private info, such as IP and data they are downloading/sharing.

In this thesis we want to provide a solution of a P2P group of untrusted nodes that could easily be integrated with a standard Content Delivery Network (CDN), as well as test its performance while sharing a mp4 file amongst nodes. We believe that a P2P group is better suited to be used in conjunction with a cloud solution rather than aiming to replace it. To do so, we used an emerging and already deployed P2P technology, InterPlanetary File System (IPFS), since it has already a HTTP API and could easily be integrated with a CDN and browser utilization.

We use IPFS [1] with a few changes in order to try to optimize it to video streaming. This is done by changing the way IPFS chooses the next peer to provide set block. Currently IPFS randomizes this choice with a bigger probability for peers that have previously provided blocks. We wanted to give a bigger probability for peers with the least latency possible.

A review of the main investigation was done to reduce CDN costs, either by using different caching methodologies or by optimizing its flow. Also, we will go through some research regarding the integration of CDNs with P2P groups and regarding the safety of CDNs and P2P file sharing systems.

To provide important background for this thesis, we describe how IPFS, the main technology used in this thesis, works. We also study in depth its modules, emphasising the ones that we changed in this thesis as well as the key points that provide an improvement when comparing with other P2P file sharing systems.

Also, we address the architecture chosen for this work explaining how the P2P works and how we provided location awareness with the changes made to the IPFS algorithm used.

As the changes made had to be tested, we explain how the tests were set up to evaluate these changes and compare them to the previous iteration of IPFS. Lastly the results were also analyzed.

Finally we take conclusions about the results and talked about the possible future work.

Chapter 2

Related Work

In this section we go through the two main research topics aiming to reduce CDNs overall cost, caching methodology (2.1) and flow optimization (2.2). We also discuss the security (2.3) research related to CDN and Edge Computing to bring security solutions to CDN cost optimization.

2.1 Caching Methodology

Cache servers aim to reduce the burden on the backhaul link to the CDN server and provide a service with less delays and buffer times therefore avoiding fetching multiple times the same video, that is otherwise stored at the cache server near the edge of the network. There is a latent trade-off between achieving the best Cache Hit Ratio (CHR) possible and the cheapest CDN infrastructure.

Auto-scalable caching [9] services reduce costs while providing the best possible performance for the CDN. By using an auto-scaling cache method with machine learning, each cache node should determine the optimal Time-To-Live (TTL) for each piece of content getting the best CHR for the best optimized cost. They do so by monitoring how many requests did each piece of content get in each cache node and then a TTL is set based on the content popularity (more requests means that a specific piece of content is more popular). This TTL value is updated according to user requests as well as the cached content itself, if a requested piece is not on the cache node, the request is forwarded to the main server caching it afterwards with an assigned TTL.

Proactive Caching [10, 11] is a method where content demand is predicted and popular content is prefetched into the cache servers. By doing so, it is possible to achieve a better CHR and less delay on first demands for the video. In [10] user patterns related to video watching are studied, so that only the watched video chunks are cached in edge servers instead of caching the entire video. In another study [11], the proactive caching approach is done by analysing content popularity through a big data platform and pre-cache it onto the edger servers.

Probabilistic cache [12] placement in wireless Device-to-Device (D2D) networks aims to maximize CHR and throughput. In this type of wireless networks, the cached content can either be stored at the network devices or the edge base stations causing this network topology to have a special case of cache hit when the requested file is within the requester itself.

Whenever a device requests a file and there is more than one device which has that file near to the requester, the closest node is the one that transmits the file. If no close node has the file, it will be transferred from the closest base station.

In this article the probabilistic cache placement refers to the probability of an occurrence of a cache hit inside the edge network with another device having the requested file.

In [13] GridCast (a P2P Video on demand (VOD) system) is explored, evaluating performance and user experience. Authors try to optimize GridCast by doing more aggressive caching and prefetching. To reduce seek latency and improve user experience, an anchor prefetching algorithm was implemented as a way to deal with random seeks.

Before this research GridCast was only locally caching the current video. This paper proposes a more aggressive approach by suggesting that multiple recently watched videos should be cached using more user resources. By doing so, the probability that a peer has the content that a user is requesting drastically increases, reducing the backhaul link burden.

Cache deployment cost optimization [14] focuses on the cache deployment process. This research focuses on optimizing cache deployment in internet core by analysing a set of metrics, such as end users traffic, and matching it with the most acceptable performance from an autonomous systems. Their Cache Deployment Operation (CaDeOp) system aims to plan the CDN deployment in a longer term enabling updates that optimize the deployment under expected future conditions.

Device Caching is used in hybrid CDN-P2P topologies leading to the use of CDN mobile nodes as a way to offload the burden on CDN and edge cache servers. This can also be a way of reducing costs by reducing the edge cache server resource requirements or potentially even completely removing it. This topologies are a CDN with different clusters where users inside a cluster share content via D2D communication whilst under cluster head coordination. Content can be cached in either the base station (BS) or the users themselves meaning that upon receiving a request, the cluster head either delivers it itself or forwards the request to another node or to the BS. This article also proposes a reward system for users who share content instead of assuming that users will do it without any kind of incentive. Lastly, authors make an analysis on the trade-offs between costs and stability of a CDN with edge caching and D2D content sharing.

Research [15] focuses on crowded environments proposing a different approach to wireless video streaming consumption and developing a secure method of D2D content sharing. Iris supports Multicast, Unicast, Retransmission, Scheduling/Multipath, Forward Error Correction (FEC), Mobile Data usage, Security features, such as authentication, authorization, forward secrecy, stream integrity and P2P while being a Full System solution. The results of the study in this article show that there are advantages to this environments.

Another popular solution regarding Device Caching providing the option for a decentralized CDN solution, is IPFS which is a completely decentralized P2P file sharing method [1]. IPFS is an already implemented solution that uses Kademlia Distributed Hash Table (DHT) overlay. In Kademlia, content is stored under a (Key,Value) pair, the Value being the content itself and the Key its hash. The pair is then stored in the nodes with the closest ID to the Key. This brings a massive draw back as we are consuming storage and bandwidth from nodes which potentially do not need that specific file and also we are ignoring the far away nodes that already had the file. Facing this challenge, IPFS uses a merge between Kademlia and Coral DSHT [16] extending Kademlia to store references to nodes which have each data block. Small sized files (1KB) are stored directly on the DHT whereas with larger files the DHT stores references to node IDs of peers which have that file and can transmit it. Additionally, it uses Merkle DAG which are a merge of merkle trees [17] and Directed Acyclic Graphs (DAG) in order to provide content addressing, tamper resistance and deduplication optimizing storage.

In [18] D2D caching and communication is explored, proposing a caching placement with mobility of users in mind.

2.2 Flow optimization in CDNs

Flow optimizations consist in defining the best routes from CDN server to edge nodes in order to achieve the best QoE possible and performance out of the CDN, while hopefully minimizing the overall cost in the process. CDN flow optimization can be done by dynamically changing traffic routes mainly exploring on the fly optimization using Software Defined Networks (SDNs) alongside CDNs to observe network status and adapt routing path and server selection. Alternatively, this dynamic routes management can also be done by managing the CDN replicas.

Another option explored is the merging of the two most popular video delivery solution, CDNs and P2P networks, with the purpose of creating a hybrid CDN-P2P network having the content cached in the edge nodes which share it with each other providing a lower delay time, better QoE and ultimately a less costly edge cache server infrastructure.

Dynamic Routing is a solution based on the implementation of SDN applications which monitor the network state and adapt traffic routes according to the network state. One author [19] makes use of MPLS to change routing paths whereas in [20] authors propose an approach where they combine both optimal server selection and optimal routing path selection by centralizing network management in a intelligent SDN platform. Moreover, an algorithm to assign user requests considering mainly response time and bandwidth to optimized simultaneously server assignment and routing path selection is proposed.

Another approach that is taken into account to optimize CDN flow is by managing its replicas. The authors of [21–23] developed ways to integrate Kubernetes fog computing with flow optimization providing the best QoE possible for the user.

Replica Management aims to optimize CDN flow as well as related costs by dynamically adding replicas in high traffic areas and removing them when few traffic occurs. Additionally, it can also be done by optimizing replica routing policies as discribed in [21] where a Kubernetes plugin, Proxy-mity, is proposed to be used in the other two solutions, that adds proximity awareness to Kubernetes traffic routing. The system administrators have to define parameter α that expresses the desired trade-off between load balancing and lower latency (closer to end user) therefore controlling the imbalance of their system. This plugin uses Serf [24] that implements Vivaldi coordinates [25] to monitor network latency between worker nodes. When Proxy-mity detects a change in the service set of pods it recomputes routing routes and injects them in Linux using iptables.

In [23] another Kubernetes plugin is proposed, Voilà. This plugin aims to provide a replica autoscaler that takes into consideration network tail-latency. By monitoring all the request workload produced by all potential traffic sources in the system, it is able to add or remove replicas as well as change their location to maintain defined application QoS. Similar to Proxy-mity, it also uses Serf to implement Vivaldi to estimate latency between nodes. Voilà also uses Proxy-mity itself to ensure that requests are received by nearby pods. The main trade-off here is between having the least possible pods to reduce the overall cost while maintaining the QoS. A round trip latency threshold is defined for each application and a request above the set threshold is flagged as slow. The number of flagged requests is used as a parameter in future replica adjustments.

Lastly, in [22] authors propose Hona, a replica scheduler which takes into consideration tail-latency. Similarly to the other two solutions, Hona also integrates with Kubernetes. The main three resources used by Hona are Vivaldi coordinates to estimate network latency between nodes, Proxy-mity to manage routing path from end-users to nearby replicas and the Kubernetes structure itself to monitor systems resources availability. After the initial placement whenever the end-user workload changes, Hona can replace replicas to meet the defined QoS metrics.

Papers [21–23] are the main components of a PhD thesis [26] that aggregates those three algorithms to develop a replica management system on fog computing platforms but with proximity awareness. This thesis integrates the algorithms together with Proxy-mity being responsible for routing the traffic for the closest replica, Hona monitors the network to be aware of any changes to be made in the replica geo-distribution to meet the QoS requirements. Both Hona and Voilà are capable of placing/replacing replicas however Voilà presents an improvement in handling load imbalance since Hona does not take replica capacity into consideration and manages the load imbalance observing deviations from standard number of requests in each replica. On the other hand, Voilà defines a metric that calculates the maximum number of requests that a replica can handle in a time span being able to deploy new replicas to avoid saturation, whereas Hona is only able to fix proximity issues and not scale up the replica set. Lastly on top of the stack is Voilà dealing with the replica set auto scaling.

Hybrid CDN-P2P is an architecture that enables the caching of content in edge nodes allowing to reduce costs in edge cache servers and providing less delay and better QoE having close peers transfer the content.

Authors of [27] propose a hybrid CDN-P2P architecture managed by edge data centers over on a SDN platform. In these solutions users are put into groups managed by the edge data centers and both CDN access and peer-to-peer video streaming between users are also managed by the edge data center of each access domain. Each access network is composed of multiple nodes connected to a multi-access edge data center.

Each edge data center has a compute node which runs the SDN controller. Some SDN apps are CDN app, responsible to manage the CDN access, and P2P app. P2P app aggregates users into group making sure the groups are formed with users near each other (location aware group forming). It also does pseudo-central chunk scheduling withing each group. The app is also responsible to manage user churn making groups dynamic. In this solution chunk scheduling is done centrally and then communicated to the peers contrary to standard P2P networks. The purpose of this procedure is to achieve a fairer network adapting to peer buffer status and incentive points as well as keeping network resource usage effective. Incentive points are a way of encouraging users to upload content.

This solution security relies on the assumption that the data channels used (WebRTC) are secure and that the sending of a video request to the CDN app pass through a authentication process.

LiveSky [28] also implements a hybrid CDN-P2P system, although, to do so it does not use a SDN. The three major components of its architecture are a Management Center (MC), cache servers and end users. The Management Center includes a DNS-based Global Server Load Balance (GSLB) system (this system is responsible by the redirection of incoming user requests to the nearest lightly loaded edge cache server), the content management and configuration system and the monitor and billing systems. The cache servers make the connection between content providers and end users. The end users can be legacy users that will only use the CDN structure obtaining the video from edge cache servers and the LiveSky users that will benefit from higher quality streams using the hybrid CDN-P2P structure.

Server nodes (SN), cache servers, are organized into tiers with the lowest tier being the closest to content source. When SN boot up, they send an alive message to the MC which then broadcasts the message to the other server nodes. In order to a SN to establish a TCP connection with other SN the first one has to require information (IP address and Port) about the second from the MC. Nodes can establish a connection with the ones on a higher or the same hierarchical level, except with edge nodes. Since these nodes are heavily loaded from user connections they can only connect with higher hierarchy nodes.

2.3 Security

While reducing CDN overall cost, it is imperative to make sure that secure CDN operations are not left out. Edge Computing technology has some inherent security concerns. An edge server must be secure from privacy leakage, denial of service, privilege escalation, service manipulation, rogue data center and physical attacks. The Edge Network may be vulnerable to denial of service attacks, rogue gateway and man-in-the-middle attacks. It is also possible for an attacker to control edge devices injecting false information onto the network in order to disrupt or control services [29].

CDN missconfiguration is studied in [30] where authors explore the failure to verify the ownership of the origin domain (a domain or IP address to which a CDN will forward requests to and pull resources from) given by the costumer. In 2018, this was a configuration option that introduced vulnerabilities in top 8 CDN providers. This failure led to a number of vulnerabilities discussed by in this article, however authors also propose mitigations to the flaws discovered. Since the main reason for this to be possible is the lack of validation for costumer-supplied information, to validate this, the user could be required to upload a specific file to the origin to verify its ownership. The constant monitoring of any changes in the origin could also help mitigating this flaw.

There has been some research focused on security in D2D caching methods. Although most of them are theoretical [31, 32], and fail to cover with practical questions such as data integrity, authenticity, confidentiality, among others. as well as secure authenticated node join. [31] focuses on a scenario where an eavesdropper tries to eavesdrop the content sharing between two nodes whereas [32] focuses on a secure caching making so no forged files are propagated between users.

Secure D2D communication is a key component to take note in hybrid CDN-P2P architectures. For a system like this to be adopted customers need to have trust in the security solutions applied.

The usage of secure tokens [15] enable communication between nodes in the edge cloud bringing Security-as-a-Service for Mobile Edge Clouds. Tokens are sent from the server to edge nodes and used to establish connection either via Bluetooth or Wifi Direct. Video chunks are digitally signed by the trusted server who is the creator of all chunks and, on its turn the packet containing the signed video chunk is then signed by the sender, ensuring data authenticity and integrity since the sender can be verified and video chunks can only be signed by the trusted server. Key Rotation is also implemented in this solution.

Secure D2D communications have to be ensured in order to have a secure D2D caching system. This type of communication does not have any central authority, for example an access point in a traditional wireless network, to manage user authentication. Peer discovery is done by broadcasting over wireless channels being susceptible to an attacker listening to the communication and tracking down user thus violation location privacy. These are some of the security challenges in D2D communication identified and addressed in [33]. The authors provide definitions to security and privacy requirements on D2D networks and show solutions on key management, authentication, confidentiality and integrity, availability and dependability, and secure routing and transmission. For each of the main solution topics, they review the existing bibliography. The same process is done for the privacy solution main topics (Access Control, Obfuscation, Anonymity, Cryptography and Application-Oriented Privacy). Finally, the author identifies the open problems that need further research.

Internet users can have their traffic eavesdropped and even if the communications are encrypted the attackers can monitor the traffic to track user patterns and infer what services, sites, etc is the user using/visiting [34, 35]. In **P2P Networks** the same can be done. Even if the nodes are anonymous and the communications between them encrypted, the information published to the DHT is still public, making it possible to know which nodes are retrieving and/or reproviding which data. Additionally, could be possible to match a node ID to an IP address, particularly if the node is always running from the same machine as demonstrated in the Bitcoin network [36]. Every node has a set of peers to which it is connected to, when an attacker, using a botnet, manages to control all the peer it is connected to, the victim node is excluded from the actual network activity. In the Eclipse attack [37] the attacker can manipulate the isolated node by sending false information. This was also tested in the Bitcoin P2P network [38]. To prevent this type of attacks, random node selection in P2P connection and an increased number of peers each node is connected to, increases the number of nodes an attacker has to control thus increasing the cost of such attack. Another popular attack in P2P networks is the Sybil attack [39] where an attacker uses a single node with many active fake identities (IDs) simultaneously and thus influence the network. To prevent this, measures like S-Kademlia [40] which adds measures to prevent attacks like this and Eclipse on the Kademlia overlay. The goal is to make node ID generation (Sybil attack) and choosing the ID freely (Eclipse attack) hard for the attacker. This is done by hashing together the IP address and port or by hashing a public key [41] to authenticate the node with his ID, since the IP can change dynamically this comes with a drawback making it better to hash the public key.

A low cost IDS/IPS [42] is proposed by implementing a dynamic software-based security system that reacts, in a dynamic and autonomous way, to observed and analysed traffic which is marked as a possible threat to protect CDN edge servers. This represents a significant improvement from existing work since previously developed solutions had a static approach to attacks such as DDoS, requiring manual intervention to be stopped or based on hardware such as traditional firewalls and IDSs. This constitutes an increase in expenses for the CDN edge server. The main goal of this research was to develop a solution with low overhead and that automatically reacts to threats recovering legitimate traffic throughput. This goal is achieved by adapting the security services overhead to the state of the CDN services, dynamically creating, modifying and removing security services. The operator specifies security policies that are used to orchestrate the security services and their response.

Secure Data Storage as a ciphertext in untrusted edge servers is a topic of research having methods like identity-based encryption [43], attribute-based encryption [44], proxy re-encryption [45] and homomorphic encryption [46] gaining traction in order to provide data confidentiality.

HoneyBot [47] is an adaptation of the traditional HoneyPot for mobile edge platforms D2D communication. HoneyBot was built to detect, track and isolate malicious device-to-device communication insider attacks. In detection phase, it logs the messages exchanged across its interfaces and analyses them searching for malicious activity. The next phase is called Tracking phase where HoneyBot tracks the source of the attack by following the path from which the malicious messages where sent. Lastly, the Isolation phase where the malicious node is physically identified and isolated from the network.

Chapter 3

Background

In this chapter we go over technologies that are directly related to this thesis like conventional CDNs and the main component of this thesis, IPFS. We explain how does it work and we will go more in depth in the modules we changed more actively.

We have chosen IPFS because, from our research, it was the best decentralized option of optimizing the cache using the network nodes and is an open source already deployed technology that was proven to work well when serving as a decentralized file system. Also, from the research made on ways to reduce CDN overall cost, using P2P seemed to be the solution with the most inpact and the least usage already in real world scenarios. There is a lot of documentation and repositories about IPFS and it has already been adopted by a respectable range of companies and other technologies.

3.1 Standard CDN

A CDN is a network of servers that are deployed in multiple geographical locations near end users in order to reduce delays when accessing content. This network of servers around the world replicate content from the origin server by caching it. By doing this replication users instead of fetching content directly from the origin server will get it from the closest CDN server. Standard CDNs try to be as close as possible to end users interacting with Internet Service Providers (ISPs). Both CDN providers (deploying their servers closer to end user) and ISPs (avoiding having to route traffic outside their network) benefits from this partnership providing better QoS and reducing costs respectively.

Good examples of widely used standard CDNs are Cloudflare and Akamai with both adding their own mechanisms to make their implementation faster and safer for their clients.

3.1.1 Cloudflare

Cloudflare works as a reverse proxy using anycast routing. This means that geo-routing is done by using the shortest path to the same IP through the common internet protocol in BGP (Border Gateway Protocol) allowing an easier and faster connection to the closest CDN server. In other words, Cloudflare servers have the same IP, so when an user connect to the IP the routing protocols selects the server that is closer. The other option would be using DNS servers to route the traffic for a close server. Using anycast Cloudflare ensures immediate failover instead of having to wait for DNS cache to reset as well as preventing DDoS attacks since the traffic would be split across the network automatically.

Static content caching is dependent on the proximity of caching servers to the end user but with dynamic content, for example a website that updates a chart with database data. This piece of information cannot be cached and has to be required from the origin server.

What Cloudflare does is it uses its own protocol called Railgun. Railgun accelerates connection between Cloudflare data centers and the origin server, speeding up any content request that cannot be served by the edge cache. It tracks changes to content down to the byte and upon request received, it only sends back the bytes that have changed. Railgun connections are secured by TLS and compressed into a binary chunk.

3.1.2 Akamai

Akamai is one of the biggest CDNs, serving companies like Airbnb, Lufthansa, Youtube (integrated with Google Cloud) and many other big brands.

It uses a standard multi tier CDN infrastructure with tens of thousands of globally deployed servers being subdivided into multiple sub delivery networks, each serving a different purpose ranging from static content caching to streaming media. Instead of deploying massive server farms in few data centers, here the approach is to decentralize as much as possible with server clusters of a variety of sizes deployed in a multitude of locations. They take this approach since they consider it is the one which provides the best efficacy in being closer to end users.

Akamai uses a mapping system that takes the URL introduced on the browser and directs the request to the edge server. This mapping system select the edge server using historical and current data regarding network and server conditions.

To provide the best streaming service possible, alongside a big number of servers distributed worldwide, Akamai uses tiered caching. Whenever an edge server does not have the content requested, instead of asking the origin server, it asks its parent cluste, which leads to the origin server only having to manage a few dozen parent clusters.

Once encoded, live streams in particular are then routed to an entrypoint, an Akamai server cluster. Additionally, copies should be sent to other entrypoints in order to create redundancy and thus avoid creating a single point of failure. The system works in a publish-subscribe model where each entrypoint publishes the streams that it has available and each edge cluster subscribes to streams that it requires. To make this process scalable, a subset of servers called reflectors are used. Reflectors work as intermediaries between entrypoints and edge server and can to receive one or more streams from the entrypoints, copy them, if necessary, and can then send the same stream or different streams to one or multiple edge clusters.

The communication between edge clusters and parent clusters are optimized using the following techniques:

Path Optimization: As stated above, Akamai's mapping system gathers internet topology and performance data. This data is then used to select intermediate nodes for a specific path. Also, Akamai can send communications over multiple paths to add resilience. Contrary to Cloudflare, Akamai does not use BGP since it often chooses sub optimal paths.

Packet Loss Reduction: By sending communications over multiple paths and applying FEC (Forward Error Correction) Akamai is able to offer significant packet loss reduction with minimal overhead and without increasing latency.

Transport Protocol Optimizations: A proprietary transport-layer protocol is used between its servers using pools of persistent connections, using optimal TCP window sizing based on knowledge of real-time network latency conditions and enabling intelligent retransmission after packet loss by leveraging network latency information, rather than relying on the standard TCP timeout and retransmission protocols.

Application optimization: Akamai uses application-level techniques to boost Web Application responsiveness for example content compression, prefetch content that might be requested in the future and so on.

The key to Akamai´s success is its highly distributed nature having its endpoints located very close to the origin server and the end user.

3.2 IPFS

Web can, sometimes, be inefficient with users having to download content from a single server. IPFS was created with efficiency in mind making users download pieces from multiple nodes enabling substantial bandwidth savings. Due to its decentralized nature, it provides better connectivity for all users as well as no single point of failure.

InterPlanetary File System (IPFS) is a Peer-to-Peer hypermedia protocol used to run a decentralized file system amongst the nodes running IPFS protocol.

IPFS gathers different well known protocols such has Kademlia, BitTorrent, Git version control and Self-Certified Filesystems (SFS) to create a fully distributed Peer-to-Peer file sharing system where nodes do not have to trust each other.

It is worth to note that IPFS paper [1] is theoretical and that some of its features are either not yet implemented or do not function exactly how it is stated.

3.2.1 How does it work

Nodes that run IPFS protocol interact with each other in order to share IPFS objects on the network. These objects are stored on its nodes local storage.

Files added to the IPFS network are split in smaller chunks. Each chunk is called a block, has a CID (Content Identifier) and each CID is unique, so it acts as a way of locating that specific chunk or any of its replicas.

It intends to be user friendly by minimising the downsides of Kademlia, such as nodes having to store files if their ID is close to the file key, therefore wasting storage on possibly unintended files and thus ignoring the nodes that already have the file. This is done by integrating Coral [16] DHT with Kademlia. With this adaptation only small files are directly stored on the nodes closer to its Key whereas for larger files only a reference to the nodes that can provide those files is stored on the nodes closer to each file Key. On larger files, node *N* first sends a ProvideValue message that propagates like a normal search for an ID, but instead tells the *K* closest nodes to that ID that *N* can provide the file with the announced Key. The closest nodes to the file Key then store a reference to each node that can provide the announced CID. Once a search is performed, these nodes will respond with the full list of providers to that Key instead of sending the requested file.

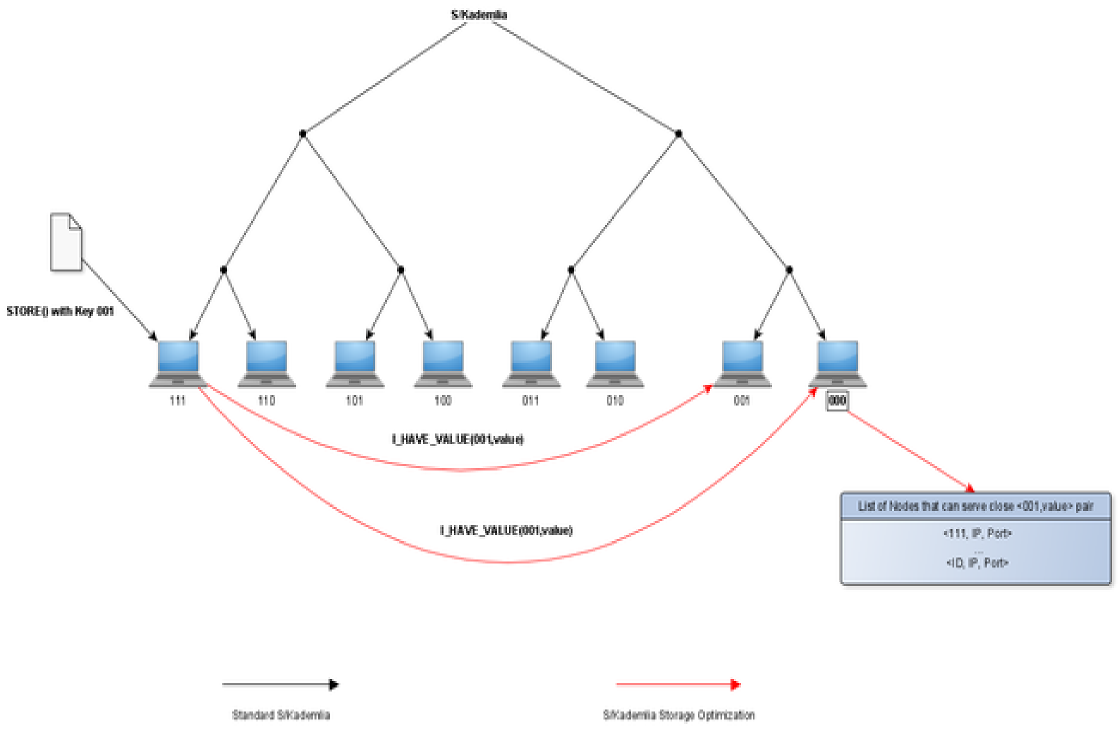


FIGURE 3.1: I_HAVE_VALUE RPC

Whenever a node searches and downloads a CID from its peer, it will also become a provider for that block until the cache gets cleared.

In case of a new version of a specific file is uploaded, its hash will be different making it resistant to tampering. However, same chunks will remain with the same CID, therefore allowing their reuse and saving storage.

IPFS has its own name system, IPNS, meaning that instead of having to remember long strings of random numbers and letters (CIDs), files have human readable names that, when searched, IPNS converts to the CID of the last version of that file. By having this system, IPFS does not search for a specific source, for example when typing a link on a browser, but rather searches for a specific object being served by a group of nodes that have such designated object.

Additionally IPFS network has a cryptocurrency attached to it, Filecoin (FIL) [48], that instead of using Proof-of-Work or Proof-of-Stake, as its consensus method, it uses Proof-of-Storage allowing nodes to pay the miners to store their objects on the network to make them available forever or just to increase their replication.

3.2.2 IPFS protocols

IPFS is subdivided into a stack of sub-protocols, Identities, Network, Routing, Exchange, Object, Files, Naming. In this section we will go over each of those sub-protocols and will emphasise the modules where we will make changes.

Identities

This sub-protocol is responsible for managing node identity generation and verification of other node identities. Each node is identified by a node ID generated by S/Kademlia which is the hash of its public key. Both public and private keys are stored and encrypted with users' passphrase. Nodes are incentivized to keep their identity to preserve network stability instead of changing it on every launch. When connecting to a peer, its ID is compared to its announced public-key. If they do not match, the connection is terminated.

Network

Network subset of protocols is responsible to establish and manage connections between peers. This subset includes many transport protocols that define which transport protocol to use in each connection, although IPFS works best with uTP or WebRTC DataChannels. This subset also provides options regarding connectivity offering NAT Traversal techniques, like ICE NAT [49], to allow peers behind a NAT to be reached. Integrity and Authenticity are also managed by the Network sub-protocols.

IPFS uses multiaddr formatted byte strings instead of conventional IP addresses. An example for a multiaddr byte string is: /ip4/10.20.30.40/sctp/1234/

In addition to the IP address version, it also points which protocols and port to be used when establishing a connection to set node.

Routing

Routing protocols encompass DHT protocols, managing information on how to find peers and peers that can serve a specific object. IPFS uses a secure DHT based on S/Kademlia and Coral making a storage distinction based on size as explained in 3.2.

Exchange

To exchange blocks, IPFS defines Bitswap, a new protocol based on BitTorrent. The changes made for this thesis are mainly done in Bitswap module, since we aim to optimize block exchange by prioritizing nodes physically closer.

BitSwap

In this subsection we go throught Bitswap in more detail since it is heavily related with the changes we propose in this thesis.

As mentioned before, Bitswap is a protocol designed to manage block exchange between nodes. Some key concepts in Bitswap are: peers, which are IPFS nodes that will actively be trading blocks with each other; a *want_list* which is a list containing information about the blocks a node wants and has, this list is shared amongst peers; and a session which is a group of peers that answered with a have message to a search broadcast. For example requests to fetch all the blocks in a file would be made within a single session.

Bitswap protocol uses a mechanism where peers send a want-have message with details about the blocks they are locking for and the blocks they own. Each peer keeps a list (*want_list*) of which blocks their peers are looking for. If they acquire any of those blocks, a message will be sent asking the peer if they want them to send that block.

To discover which peers have a block/file, Bitswap broadcasts a want-have message to all peers it is connected to asking if they have the CID for that block or the CID root for that file. Any peers that have the block respond with a have message and are added to a session.

Then it sends a want-block message to a pseudo random peer in the session. On the other hand, if the peer does not have the block, it responds with a *dont_have* message and Bitswap send another want-block message to another peer.

This selection is pseudo random because peers that have served blocks before have a higher chance of being selected.

If no peers in the session have a specific block, the node queries the peers it is connected to. If none of them have the block, Bitswap queries the DHT to find providers for that block.

Bitswap block transfers could be recorded on the Filecoin ledger creating a system where blocks would be paid for. This is not implemented, however the option is presented in IPFS paper [1] and the Filecoin ledger is already deployed. Alternatively, a ledger is

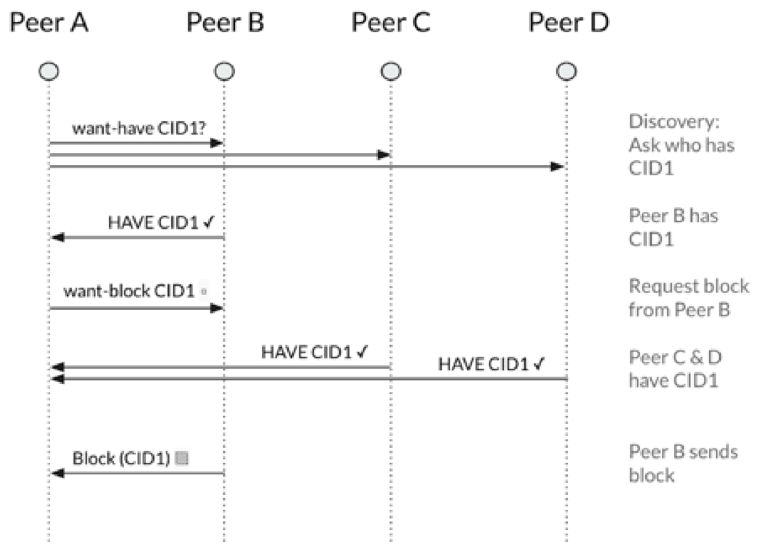


FIGURE 3.2: BitSwap - Discovery

kept on Bitswap recording block transactions and creating the knowledge of how much each block were asked and served.

Going deeper on how the code works, IPFS is coded mainly in Golang and JS. For this work we used the Go version.

Bitswap module uses as its arguments a Bitswap network, a blockstore and a set of options to initialize the block exchange. Once initialized, it spawns workers to handle incoming requests from other nodes, send out provide messages and deal with incoming blocks.

Whenever Bitswap receives a request from another node, it is handled and then stored on a *PeerRequestQueue*, pulled by the engine and then passed to the workers to forward it. Anytime a IPFS user requests a file a session is created with peers that can provide blocks for that file. Each session has: a *sessionWants* that keeps track of what *Wants* were sent but not yet received; the order they were sent and a queue for the next *Wants* being sent; a *latencyTracker* that tracks the average latency between sending a want a receiving the response; a *sessionWantSender* who´s responsible for choosing which peer to send a Want to; and a *PeerManager* to manage the peers in a session.

On a lower level, messages are sent using Bitswap Network and allowing for latency to be measured. A message can contain a block, a *want_list*, *Haves* and some control information.

Object

Object sub-protocol uses Merkle DAG (Directed Acyclic Graphs) [50] of content-addressed to represent arbitrary data structures following Git data structures and linking object in the network with cryptographic hashes.

This data structure system allows IPFS to have content uniquely identified by its multihash checksum making it tamper resistant, since all content can be verified against its checksum. Since objects with the exact same multihash are equal only one should be stored preventing content duplication and waste of storage. For example, a path to an object should look like this:

/ipfs/<hash-of-object>/<name-path-to-object>

The ipfs prefix works as a mounting point on an existing system. The second part of the path is the hash of the object. Since there is no root object (that would imply the impossible task of handling consistency of millions of objects in a distributed environment), the root is simulated with content addressing, meaning that every object can be accessed by its hash. In the example bellow, the last object (baz) can be accessed by the ones above it.

/ipfs/<hash-of-foo>/bar/baz

/ipfs/<hash-of-bar>/baz

/ipfs/<hash-of-baz>

Files

Files is a set of protocols inspired on Git to define a versioned hierarchical file system model:

1. block: a variable-size block of data.
2. list: a collection of blocks or other lists.
3. tree: a collection of blocks, lists, or other trees.
4. commit: a snapshot in the version history of a tree.

Naming is a self-certifying mutable name system that works almost as a DNS.

Every user is assigned a namespace at:

/ipns/<NodeId>

The user can then publish an object to his namespace by signing the path with his private key, therefore, when another user retrieves the object, he can verify if the signature matches the public key and node ID.

This is called InterPlanetary NameSpace (IPNS) and is used to assigned mutable paths whereas IPFS has immutable paths.

Long array of random bytes is not the most user friendly way of calling an object. To counter that, IPNS allows nodes to link peer files in their own namespace and shortens the link like DNS and Urls do.

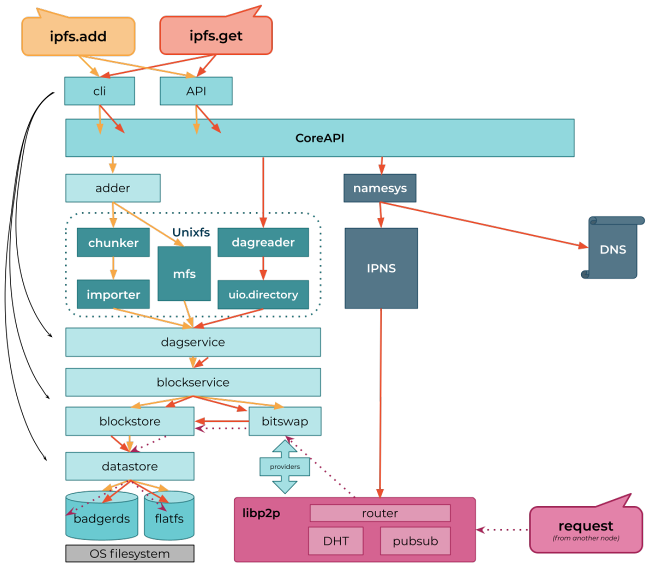


FIGURE 3.3: IPFS Workflow

Figure 3.3 shows the interaction between IPFS modules. We will work directly on Bitswap and its interaction with the DHT, blockstore and the providers interface.

All these protocols come together to form a distributed file sharing system. The first priority of IPFS is to be reliable and user friendly. Since it is designed to be a file system, delays are not the main priority. With this thesis we want to shift that priority to provide a version of IPFS capable of being integrated with a normal CDN to provide for example video streaming while providing the required QoE for the end user.

3.3 IPFS Privacy

To share files, nodes have to publicly advertise the CIDs of the files they have or want through a Distributed Hash Table. Making so that the metadata published to the DHT associated with the exchange of each block is public, despite the communication between each node being encrypted.

IPFS developers argue that key principles of the protocol's highly modular design require it not to have a built-in privacy layer, so that different uses can have different privacy solutions.

Some suggested additional measures to ensure more privacy are disabling reproviding, encrypting sensitive content or running a private IPFS network.

Node Identifiers are public, such as the CIDs, and therefore it is possible to make a DHT lookup for a specific node ID. If consistently running the node from the same physical location, the IP address can be found. Although being possible to change and reset the node ID, it comes with extra cost for the network.

With CIDs and Node IDs being public and with long term monitoring of the network, it is possible to gather information on which blocks is a specific IP requesting, sending and when.

3.3.1 Encryption

IPFS uses transport encryption assuring that anyone eavesdropping cannot read what is being sent from one node to another, whereas it does not encrypt the content itself causing that a hacker that accesses anyone's IPFS node can read stored files in both cache and system. Moreover, anyone that has a specific file CID can download it through IPFS as long as it is available on any network node.

Once again, this is an option by IPFS developer to keep it lightweight and allow developers to choose the best encryption method for their project.

3.3.2 Privacy Enhancements

As stated above, IPFS developers took a modular approach meaning that privacy was, by default, left aside. Although depending on the usage intended, some suggestions are made to enhance privacy either by changing the IPFS config file or making some changes on top of IPFS.

Share Control

By default, every IPFS node is a provider, meaning that every block received is able to be reprovided to other nodes. This, however, can be changed in IPFS config file to only announce itself as a provider for pinned CIDs that are in its cache. This way, the node can be used to provide for content that it cares about and that it wants to continue to be available on IPFS.

Using this, you have to make sure to specifically pin the content you want to reprovide and keep in your node cache.

Public Gateway

Public IPFS gateway is a ”bridge” that allows users to request IPFS-hosted content via HTTP. With this method, the gateway is participating in the network on your behalf allowing you to get IPFS-hosted content without revealing any information about your local node.

Actually, it is not that advantageous. Using a gateway will introduce delay and DHT users will still be able to see content requested through a gateway, but not be capable of knowing who requested it.

Trusting the gateway operator is key here, since technically they could be collecting data, such as tracking the IP addresses that use a gateway, and correlating those with what CIDs are requested.

Another way to proxy your IPFS traffic could be running IPFS over Tor. Comparing with the public gateway this method will avoid the trust issue with the operator but will add significant delay. If your application is not delay sensitive, then this should be a great option.

Encrypted files

To mitigate the fact the IPFS does not encrypt content as explained above, one quick fix would be to encrypt the content before adding it to the IPFS network.

This in conjunction with the usage of IPFS over Tor would ensure that your identity and file content are both kept private at the expense of some additional delay.

Private Network

As described in section 4.1, it is possible to create a private IPFS network providing full protection against public monitoring without any added measures.

Only nodes that have the same swarm key are allowed to take part in that network. However, this can lack the scale benefits such as geographic resiliency and speedy retrieval of high-demand content. It can be a great alternative for corporate implementations of IPFS allowing each company to define the intended network topology.

Chapter 4

Architecture

In this section we go over our proposed architecture and the changes made to IPFS-Bitswap. The scope of this work is to build a system capable to deliver a more cost-efficient video delivery service. We suggest this approach with the intent that it be applied either as a full hybrid P2P-CDN multi-tier infrastructure or just as a P2P group with tools for the content owner to control and grow its distribution.

It can be used in conjunction with a full institutional multi-tier fog CDN structure, with only the cloud structure and P2P group or by only deploying the P2P group.

In the multi-tier hybrid CDN approach preferably the user gets the desired content from the closest peers. If it is not available in the P2P group a request is sent to the closest edge server. Lastly if no other option is suited it will be served by the cloud CDN or by the closest edge server (after being served by the cloud CDN).

We want to maintain each user′s rights to engage as a provider in the network in either the approaches. Although, making this choice could come with drawbacks. Either they can be monetary, with companies deciding to give financial bonuses to users who opt to provide the whole network with their blocks, or in the form of drawbacks in latency and priority when being served by the other peers on the network. This way it is possible to give the best QoE to users who provide blocks. This financial study and incentives given to the user is not the scope of this thesis and is set as a future work.

For this, as explained above, we used IPFS as our baseline and modified its BitSwap module to further optimize the block exchange for video consumption and record in a database the interactions between nodes so that we can later analyze the results and take a conclusion.

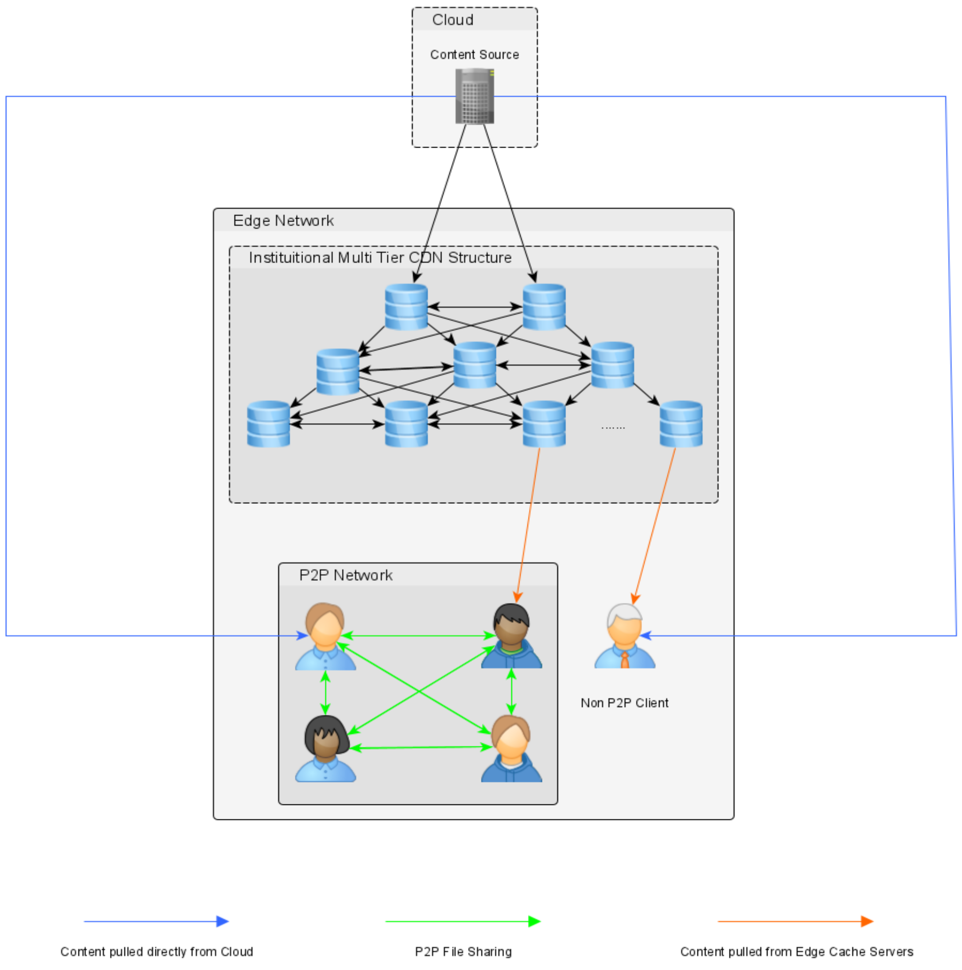

Cloud
Content Source
Edge Network
Instituitional Multi Tier CDN Structure
....
P2P Network
Non P2P Client
Content pulled directly from Cloud
P2P File Sharing
Content pulled from Edge Cache Servers

FIGURE 4.1: CDN architecture with a P2P group

4.1 Private IPFS network

In order to build a private IPFS network, after cloning our IPFS Github repository and installing IPFS, we generated a swarm key that was saved to ".ipfs" folder and copied to every node that we wanted in this private network.

Secondly, we removed all the default bootstrap addresses and added only the address of our bootstrap node and we set an environment variable LIBP2P_FORCE_PNET to 1.

Lastly, in the config folder under the address section the API address is set to localhost by default. To enable our nodes to "see" each other, we needed to change this to each node private IP since Google cloud nodes on the same account have access to each other's private ip. To emulate this in a real-world scenario where almost every node is behind a NAT router we would have to change this *localhost* to every node public IP.

4.2 P2P

This subsection is dedicated to explain how the P2P group works. IPFS uses Kademlia which is the most broadly used P2P overlay and provides fast look ups with a low maintenance cost. In this work we aim to provide the best QoE possible at the lowest cost and for that Kademlia content replication and storage brings a disadvantage.

In standard Kademlia each file is stored as a (key,value) pair in nodes which have a closer *NodeID* to the key of such file. This not only brings the disadvantage of having nodes waste storage with a not requested file but also potentially causing searches to hit faraway nodes when could have requested the file from geographically closer peers. The objective of this work is to take advantage of nodes that already had a file because they needed it before rather than having nodes store content for the sake of replication. IPFS already provides optimization in this part by integrating Kademlia with Coral for his DHT as explained in section 3.2.1.

Whenever an IPFS node receives a block, it sends a message to the nodes with IDs closer to the received file Key announcing it is ready to provide that block. Those nodes, then, add the Node ID of the block that sent them a provide message to the list of providers for that block.

When an IPFS node wants a block, Bitswap is used as the protocol that manages block-/file exchanges. Bitswap will then query the DHT for nodes that can provide that specific block/file. With the list of providers, Bitswap connects to those providers and if they respond affirmatively to the *want_have* RPC then adds them to a session. The method of choosing a peer to serve a block inside a session is pseudo random. However, peers that have previously served more blocks are more likely to be selected.

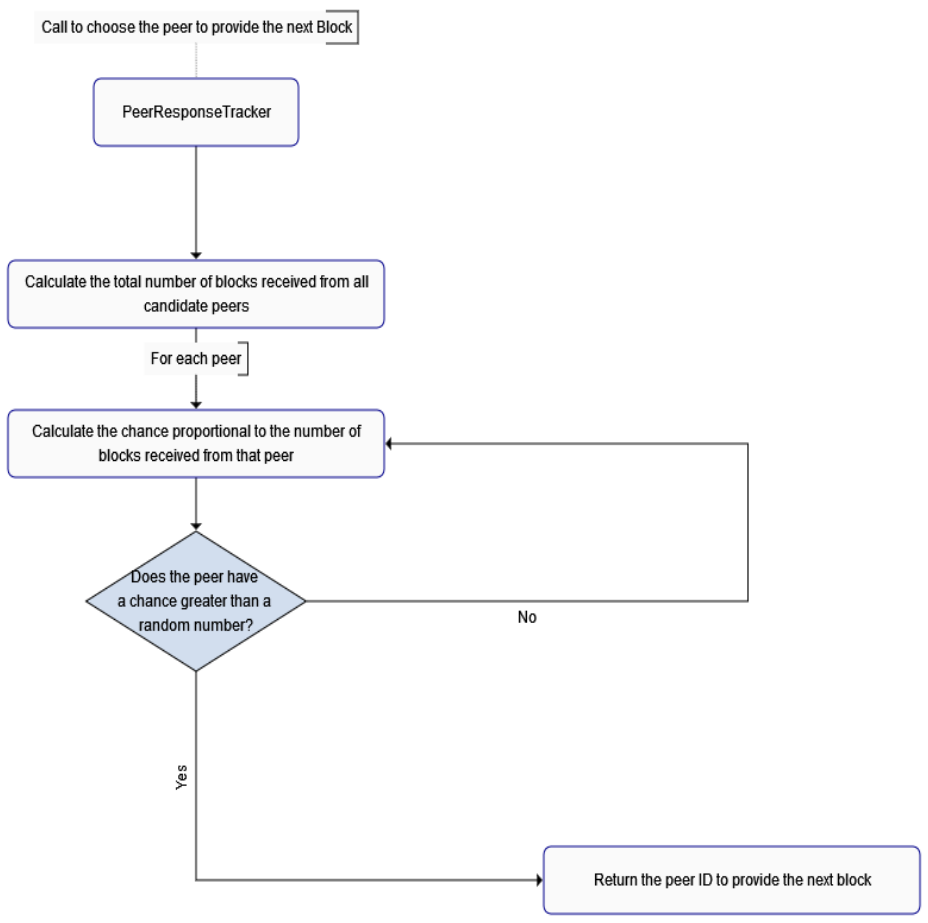

Call to choose the peer to provide the next Block
PeerResponseTracker
Calculate the total number of blocks received from all candidate peers
For each peer
Calculate the chance proportional to the number of blocks received from that peer
Does the peer have a chance greater than a random number?
No
Yes
Return the peer ID to provide the next block

FIGURE 4.2: Default Bitswap Flow diagram

Algorithm 1 Choose Peer To Serve Block

```
1:  procedure CHOOSE                          ▷ Input: peers          ▷ Output: peer
2:      $peers \leftarrow array$ of peer IDs
3:      $peer \leftarrow$ ID of the chosen peer
4:      $rnd \leftarrow$ Random $Float64$
5:      $total \leftarrow 0$
6:      $counted \leftarrow 0.0$
7:      if $len(peers) = 0$ then return $\emptyset$
8:      for every p in peers do
9:          $total \leftarrow total + blocksServed(p).$
10:     for every p in peers do
11:         $counted \leftarrow counted + Float64(blocksServed(p)/Float64(total)).$
12:         if $counted > rnd$ then return $p$
```

Algorithm 1 depicts the Bitswap behavior when choosing a peer to serve a block. It takes a list of peers as input, for each peer in the list calculates the blocks served before adding it to a total of blocks served by those peers in the list. With this information, it then selects the peers to serve the next block in a pseudo random way where the peers that have served more blocks before have a higher chance of being chosen.

4.2.1 Location Awareness

To bring Location Awareness to our P2P mesh we had two options.

The first one being very light was to alter the way IPFS bitswap protocol selects which peers would send us the required blocks. Default Bitswap prioritizes peers in the session which had previously sent blocks. Our idea was to prioritize peers with the least latency, maintaining still a small amount of randomness to this process to avoid overloading the closest peer and to maintain the core logic of Bitswap choice process.

Since we believe that every provider in our implementation will be able to send their blocks whenever necessary, we are able to make this change. As a result, the system will benefit from a slight optimization in this area since blocks will likely be sent from the physically closer peers in the session, reducing latency and consequently delays.

Because of how IPFS operates, and since we are focusing in video consumption, the blocks requested will exceed the size limit defined by IPFS for block to be stored in the peer with closer ID to the blocks Key. Therefore when requesting a block a list of all the providers will be returned making it possible to chose the physically closer peers.

We first thought that when queried the DHT would return only a small number of providers for a block. With that in mind we thought of a second solution taking a more passive approach where each node measures the latency with peers whenever a communication takes place and stores the K peers with the least latency in an extra bucket. This bucket would be updated when a connection received from a closer node than the ones on the list or if any node on the list did not reply to the regular PING messages. In that case it would be dropped from the list. In short would be maintained as a normal Kademlia bucket.

Peers would be added both to the regular Kademlia buckets and to this extra bucket, in case it is a peer with lower latency than the ones already in there.

Generally speaking, we intended to add another KBucket on top of Kademlia to keep track of the K physically closer nodes.

The point of this extra bucket would be to prioritize these nodes over the regular DHT search. This could be used, for example, in conjunction with Bloom Filters [51] to optimize the search process in the closer nodes.

However, since we IPFS sends all the providers for a block this second solution becomes redundant over the first one and probably less efficient for the extra cost of maintaining that bucket and search effort if there was no node able to provide from the K closer nodes bucket.

Following this train of thought, we decided to implement Bitswap with 4 modes of operation to compare them and analyze the results. These modes of operation are defined using the IPFS options file. We have changed Bitswap options to accommodate a provider selection mode, to select how Bitswap chooses next peer to provide a specific block, a server address that is used to tell Bitswap where to send the logs of operation to and an Integer to define the threshold for a session latency average.

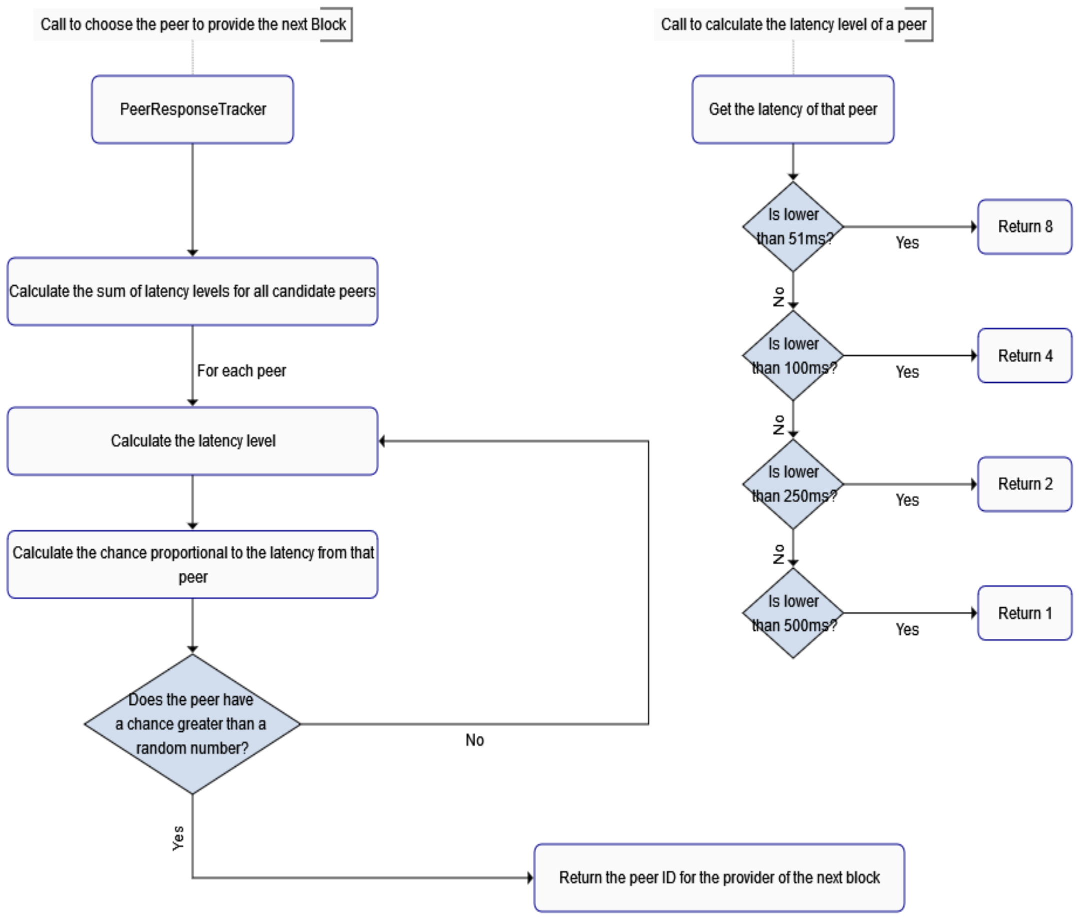

Call to choose the peer to provide the next Block
PeerResponseTracker
Calculate the sum of latency levels for all candidate peers
For each peer
Calculate the latency level
Calculate the chance proportional to the latency from that peer
Does the peer have a chance greater than a random number?
No
Yes
Return the peer ID for the provider of the next block
Call to calculate the latency level of a peer
Get the latency of that peer
Is lower than 51ms?
Yes
Return 8
No
Is lower than 100ms?
Yes
Return 4
No
Is lower than 250ms?
Yes
Return 2
No
Is lower than 500ms?
Yes
Return 1

FIGURE 4.3: Bitswap Mode 2 Flow diagram

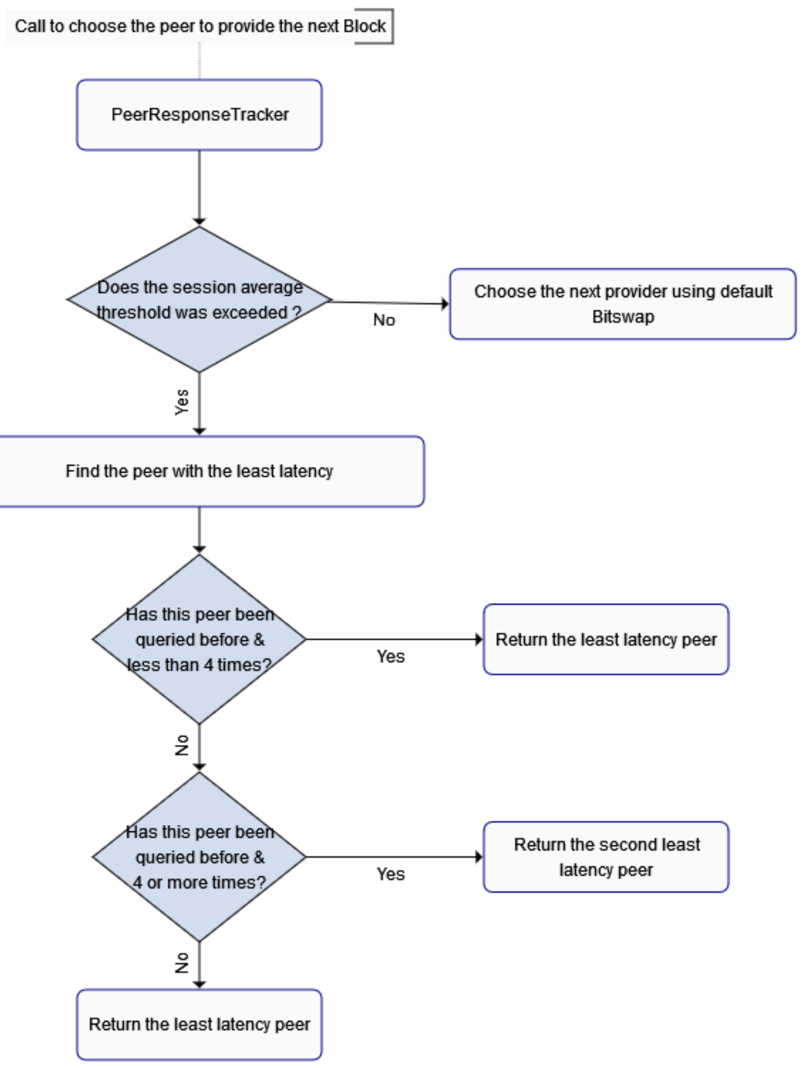

Call to choose the peer to provide the next Block
PeerResponseTracker
Does the session average threshold was exceeded ?
No
Choose the next provider using default Bitswap
Yes
Find the peer with the least latency
Has this peer been queried before & less than 4 times?
Yes
Return the least latency peer
No
Has this peer been queried before & 4 or more times?
Yes
Return the second least latency peer
No
Return the least latency peer

FIGURE 4.4: Bitswap Mode 3 Flow diagram

Figure 4.3 depicts the flow of Bitswap second mode with the changes implemented by us. Figure 4.4 depicts Bitswap third mode which is similar to the fourth. The fourth mode, however, uses the second mode core logic to choose the next provider whenever the session latency threshold is not reached.

The default mode of operation is 1, and it functions as Bitswap did before we made our changes. The second mode of operation instead of prioritizing peers that have served more blocks before prioritizes based on their latency. The thresholds defined are below 50ms, below 100ms, below 250ms, below 500ms are based on [52]. The exponential growth intends to give a much higher priority if the peer has a lower latency. If any peer has no latency measures yet, it will be given the least priority level. We use the same pseudo random method used by Default in Bitswap but instead of adding up the total blocks served by the list of peers we attribute each peer a priority Integer according to their latency, we sum it up and then use this sum to choose a peer with a chance proportional to its latency level.

The third and fourth modes of operation will act like the first and second respectively with a little twist. Whenever a session receives blocks from a peer, the *handleReceive* method is called and the total latency for fetching each block is measured. From there we calculate the average latency in that session and, if it is above our session threshold defined in the config file, the session calls a function on its *sessionWantSender* changing the *sessionAvgLatThreshold* boolean to true. Whenever the choose method from *peerresponsetracker* is called, it checks whether the *sessionAvgLatThreshold* is set to true or false. If it is true, and either the third or fourth modes of operation is set in the config file, it will prioritize the least possible latency peer for the maximum of 4 blocks in a row and then switch to the next least latency peer. We switch to the second least latency peer after the fourth successive query to avoid overloading the same peer over and over, as well as optimize the block fetching if done concurrently. This way, we can reduce the session average latency with a more aggressive approach and if it goes bellow our average threshold again it will set *sessionAvgLatThreshold* to false returning to the normal operating mode which is the mode used in first and second modes respectively for third and fourth modes.

The logic behind Bitswap modes 3 and 4 could be used in a hybrid P2P CDN to manage when the peers should request blocks from the edge servers and when to request them from peers.

The Algorithm 2 illustrates the changes made to the choose procedure, the choose procedure operates according to the operation mode and the *chooseClosest* and *getLatencyCount* procedures are used as for operation modes 2 and 4.

Algorithm 2 Choose Peer To Serve Block

```
1: procedure CHOOSE        ▷ Input: peers, sessionAvgLatThreshold        ▷ Output: peer
2:     peers ← array of peer IDs
3:     sessionAvgLatThreshold ← bool True if Threashold was reached false if not
4:     peer ← ID of the chosen peer
5:     providerSMode ← int identifies BitSwap operation mode
6:     rnd ← Random Float64
7:     total ← 0
```

```
 8:     $counted \leftarrow 0.0$
 9:     if $providerSMode = 2$ then return chooseClosest(peers)
10:     else if $providerSMode = 3$ OR $providerSMode = 4$ then
11:         if $sessionAvgLatThreshold = true$ then
12:             $closestPeer \leftarrow leastLatencyPeer(peers)$
13:             if $closestPeerQueried = closestPeer$ & $successiveQueries < 4$ then
14:                 $successiveQueries \leftarrow successiveQueries + 1$
15:                 return $closestPeer$
16:             else if $closestPeerQueried = closestPeer$ & $successiveQueries >= 4$ then
17:                 $successiveQueries \leftarrow 1$
18:                 $closestPeerQueried \leftarrow nextLeastLatencyPeer(peers, closestPeer)$
19:                 $return \leftarrow closestPeerQueried$
20:             else
21:                 $closestPeerQueried \leftarrow closestPeer$
22:                 $successiveQueries \leftarrow 1$
23:                 $return \leftarrow closestPeer$
24:         else if $providerSMode = 4$ then
25:             $return \leftarrow chooseClosest(peers)$
26:         else
27:             $return \leftarrow chooseDefault(peers)$
28:     else
29:         $return \leftarrow chooseDefault(peers)$

30: procedure CHOOSECLOSEST                ▷ Input: peers                ▷ Output: peer
31:     $peers \leftarrow array$ of peer IDs
32:     $peer \leftarrow$ ID of the chosen peer
33:     $rnd \leftarrow$ Random $Float64$
34:     $total \leftarrow 0$
35:     $counted \leftarrow 0.0$
36:     if $len(peers) = 0$ then return $\emptyset$
37:     $j \leftarrow patlen$
38:     for every p in peers do
```

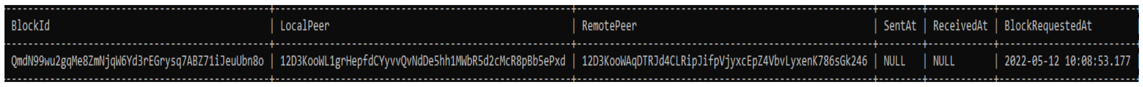
```
39:         $total \leftarrow total + getLatencyCount(p)$.
40:     for every p in peers do
41:         $counted \leftarrow counted + Float64(getLatencyCount(p)/Float64(total))$.
42:         if $counted > rnd$ then return $p$

43: procedure GETLATENCYCOUNT                ▷ Input: peer          ▷ Output: int
44:     $peer \leftarrow$ ID of a peer
45:     $latency \leftarrow$ Latency(peer)
46:     $bottomLevel \leftarrow$ 50ms
47:     $middleLevel \leftarrow$ 100ms
48:     $topLevel \leftarrow$ 250ms
49:     $avoidLevel \leftarrow$ 500ms
50:     if $latency < bottomLevel$ then return 8
51:     if $latency < middleLevel$ then return 4
52:     if $latency < topLevel$ then return 2
53:     if $latency < avoidLevel$ then return 1
        return 1
```

4.3 Log Recording

For us to be able to later evaluate the impact of set changes we needed to record the sending and receiving of blocks as well as their timestamps to later calculate the latency between when the block was asked and then sent and between when the block was send then received. We set up a GRPC server in Golang that is going to receive GRPC calls from a worker inside Bitswap. For this change to have the least impact possible, the worker is running on its own goroutine receiving information through a go channel. The information passed through this channel contains a rpcType that identifies the type of operation that we want to save regarding the blocks exchanged by IPFS peers. These operations can refer to a block *Want*, a *Receive*, a *Send* or an *Over*. The rpcType Over is only received when the IPFS command export is executed, meaning that the test is over, and it is safe to export all the gathered operations to the database without compromising IPFS performance.

Upon exporting the data to the GRPC server, the server then saves the logs in the database with every database entry having information regarding the CID of the block which it refers to, the ID of the peer sending the message, the ID of the peer receiving it and finally a timestamp referring to the moment where the *Want* was sent, the block was sent or the block was received depending on the type or message we are logging.

Each database entry looks like this:



```
---------------------------------------------+----------------------------------------------------+----------------------------------------------------+--------+------------+-------------------------
BlockId                                      | LocalPeer                                          | RemotePeer                                         | SentAt | ReceivedAt | BlockRequestedAt
---------------------------------------------+----------------------------------------------------+----------------------------------------------------+--------+------------+-------------------------
QmdN99wu2gqMe8ZmNjqW6Yd3rEGrysq7ABZ71iJeuUbn8o | 12D3KooWL1grHepfdCYyvvQvNdDe5hh1MWbR5d2cMcR8pBb5ePxd | 12D3KooWAqDTRJd4CLRipJifpVjyxcEpZ4VbvLyxenK786sGk246 | NULL   | NULL       | 2022-05-12 10:08:53.177
---------------------------------------------+----------------------------------------------------+----------------------------------------------------+--------+------------+-------------------------
```

FIGURE 4.5: Database entry example

4.4 Export Command

In order to export the logs from each node to the database we first had every node export the logs during the execution of the get file command. Although this was done in a separate goroutine, after the first testing it was clear to us that something was delaying the retrieving of the file. After some debugging, we found out that for some reason, either due to the disk writing capacity on the server VM or due to the number of GRPC connections made to the server, this sending of rpc calls by the nodes to the server was massively delaying the acquisition of the file by every node.

To keep it simple and guarantee that sending the logs to the database did not affect IPFS performance during the tests, this process was separated from the acquisition of blocks.

To do this, we kept sending the info about each block transfer to the rpc worker, on a separate goroutine, through a go channel, but instead of immediately processing it and sending it to the server/database this information was kept in memory on an array.

To export this data to the database we created a new command on IPFS, called *export*. This command, when issued, sends a ”message” to our rpc worker through the open go channel signaling that the test has ended and that it is safe to export the saved logs to the database.

This way we made sure to export the logs with no interference to the acquisition of block during the get file command.

Chapter 5

Test results and analysis

5.1 Setup

To test the proposed implementation of IPFS, we used Google Cloud VMs to create a private IPFS network, first with ten nodes then with fifty. Alongside the IPFS nodes we also had two additional VMs, one serving as bootstrap for this network and another as the server/database to receive and save the information about the blocks being exchanged.

Both the bootstrap and server VMs were running in the European data center whereas the network nodes were generated randomly with a python script that for every node being created would randomly pick a data center from the ones available on Google Cloud Compute Engine. These nodes were created based on a snapshot with our version of IPFS and a swarm key installed. Then, using a shell script run through ssh from a python script on the recently created VM, IPFS was initialized, booted and the config file was changed to have the correct bootstrap and server address so that each node could join our private network and send the logging info recorded during Bitswap execution to our server.

After setting up the nodes properly and after running the IPFS daemon on all of them, all the nodes which did not had the file, sent a request at the same time. When all the nodes had acquired the file, the export command, introduced by us on IPFS, was executed on all the machines in order to safely export the logs to the database without affecting the IPFS performance.

5.1.1 VM control

The interaction with Google Compute Engine was made through python scripting using Apache Libcloud library to manage VMs and ssh to run shell scripts on the VMs.

We used threading to assure the scripts were run in all the VMs as closely as at the same time as possible, we used python scripts to create new VMs and to run shell scritps on all the VM which included: starting an IPFS daemon; getting a file; exporting the block exchange logs and changing IPFS config file. For this last one, and since the config file was a JSON, we used jq which is a command-line JSON processor that was already pre-installed on the snapshot we used to create the VMs.

5.1.2 Test Scenarios

Our testing consisted in three scenarios where nodes would share a mp4 file with 339MB. For each scenario we did two tests, one for each mode of Bitswap being tested (Default and our implementation).

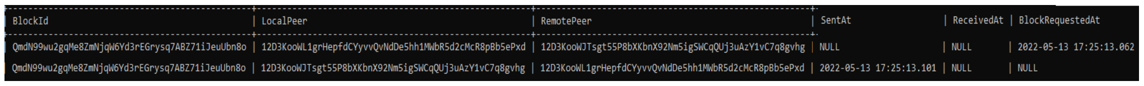
The first scenario had ten nodes where one of them had the file and the other nine would request it at the same time. The second scenario had fifty nodes, with only one having the file and all the other ones requesting it at the same time. Lastly, the third scenario was done with fifty nodes once more but this time with four nodes owning the file. These nodes were spread across the different Google Compute Engine data centers namely one in Europe, one in the United States, one in Asia and one in Australia. Again, all the nodes, rather than these four, would request the file at the same time. This scenario had the goal to, in a small scale, emulate a real-world utilization of IPFS where near each region a set of nodes would have a specific file and the other ones would be requesting it.

In every scenario nodes were using the same setup between tests. This way we made sure that location was kept the same for every node and that we did not add another variable by using different node locations. In each scenario two tests were made, one using Bitswap mode 1 with the default IPFS solution and a second using mode 2 of Bitswap with our solution. Between tests the IPFS cache and repository were cleared as well as our database.

In section 4.2.1 we proposed two additional Bitswap modes, however, when testing these modes with the first scenario we quickly realized that these would massively decrease performance as further explained in the end of section 5.2 below.

5.1.3 Data Processing

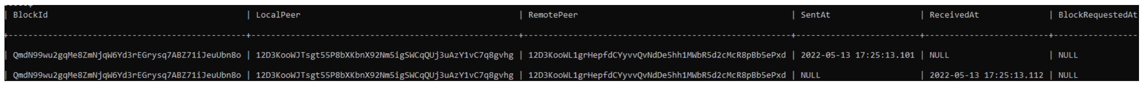
To analyze the results obtained in each test, we exported the database to a backup SQL file and ran a python script to cross the database entries with each other and find matching entries for each block and thus get the delay between requesting and sending a block and between sending a receiving a block. Two matching entries consist of two entries with the same BlockId and a relation between LocalPeer and RemotePeer. For example, when looking for a match to generate the delay between requesting and sending a block, a match would be two entries with the same BlockId and inverse match in LocalPeer and RemotePeer with one entry having the RequestedAt timestamp different from NULL and the other the SentAt timestamp as shown in figure 5.1.



```
+----------------------------------------------+-----------------------------------------------------+-----------------------------------------------------+
| BlockId                                      | LocalPeer                                           | RemotePeer                                          | SentAt                  | ReceivedAt | BlockRequestedAt
+----------------------------------------------+-----------------------------------------------------+-----------------------------------------------------+
| QmdN99wu2gqMe8ZmNjqW6Yd3rEGrysq7ABZ71iJeuUbn8o | 12D3KooWL1grHepfdCYyvvQvNdDe5hh1MWbR5d2cMcR8pBb5ePxd | 12D3KooWJTsgt55P8bXKbnX92Nm5igSWCqQUj3uAzY1vC7q8gvhg | NULL                    | NULL       | 2022-05-13 17:25:13.062
| QmdN99wu2gqMe8ZmNjqW6Yd3rEGrysq7ABZ71iJeuUbn8o | 12D3KooWJTsgt55P8bXKbnX92Nm5igSWCqQUj3uAzY1vC7q8gvhg | 12D3KooWL1grHepfdCYyvvQvNdDe5hh1MWbR5d2cMcR8pBb5ePxd | 2022-05-13 17:25:13.101 | NULL       | NULL
```

FIGURE 5.1: Entry match to calculate delay between requesting and sending a block

Else, if we are looking for a match to generate the delay between sending and receiving a block we need two entries that have the same BlockId, same LocalPeer and same RemotePeer but one of them has the SentAt timestamp different from NULL and the other has the ReceivedAt timestamp as shown in figure 5.2.



```
| BlockId                                      | LocalPeer                                           | RemotePeer                                          | SentAt                  | ReceivedAt              | BlockRequestedAt
+----------------------------------------------+-----------------------------------------------------+-----------------------------------------------------+-------------------------+-------------------------+----------------
| QmdN99wu2gqMe8ZmNjqW6Yd3rEGrysq7ABZ71iJeuUbn8o | 12D3KooWJTsgt55P8bXKbnX92Nm5igSWCqQUj3uAzY1vC7q8gvhg | 12D3KooWL1grHepfdCYyvvQvNdDe5hh1MWbR5d2cMcR8pBb5ePxd | 2022-05-13 17:25:13.101 | NULL                    | NULL
| QmdN99wu2gqMe8ZmNjqW6Yd3rEGrysq7ABZ71iJeuUbn8o | 12D3KooWJTsgt55P8bXKbnX92Nm5igSWCqQUj3uAzY1vC7q8gvhg | 12D3KooWL1grHepfdCYyvvQvNdDe5hh1MWbR5d2cMcR8pBb5ePxd | NULL                    | 2022-05-13 17:25:13.112 | NULL
```

FIGURE 5.2: Entry match to calculate delay between sending and receiving a block

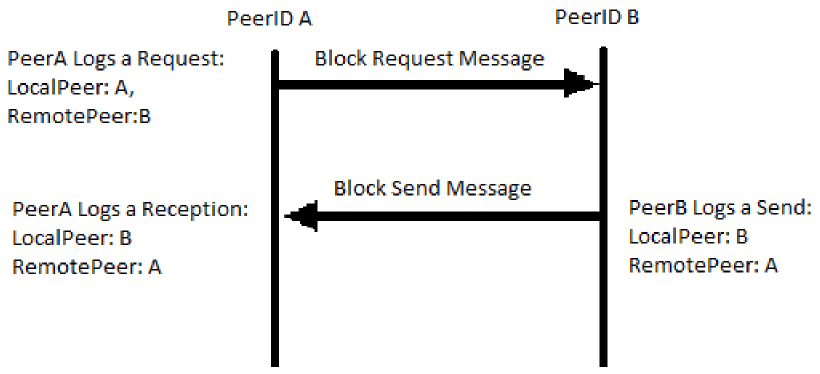


FIGURE 5.3: Logging process on each message

This entry matching is done because when saving the information about block transfers during Bitswap execution the LocalPeer ID corresponds to the node sending the message and the RemotePeer ID to the one receiving it, therefore when saving a transmission and a reception of a block, the message is the same thus the LocalPeer and RemotePeer will also be the same like shown in figure 5.3.

During this process the script is saving the analyzed results to four arrays: one for request delays; one for send delays; one for duplicate requests and one for duplicate sends. Every time a new match is found, the script checks if there is already an entry with the same peers and Block Id either in the requests array or sends array depending on what match it is looking for. If there is, then proceeds to check which of the two was first and saves the second to the respective duplicates array. For the sends delay array, specifically, it also checks if the receiver has received the same block from other peers and if it has, then checks which peer was first to deliver the block thus saving the other results to the sends duplicate array.

After crossing every database entry with all the other entries and with all the information gathered, the data is stored temporarily in arrays is then saved to 4 excel files, one for each array, with 5 columns (BlockID, Sender ID, Receiver ID, Timestamp, Lantecy).

Lastly, another python script is run to generate another excel file with the requests that actually were sent to the block provider. This is done by checking for rows in the recently generated requests delay excel file that match the ones on sends delays excel file. By doing, this we get a better grasp on the request delay.

5.2 Result Analysis

To analyze our results, we compared the average request and send delays, the standard deviation and the number of duplicate blocks being sent for both tests in each scenario. To lower the amplitude, we used confidence interval at 95%.

More duplicates mean more network overhead and a longer delay means that the blocks take longer to arrive.

In the first scenario there is a very slim margin in every metric apart from the duplicate blocks being sent which shows a decrease of 16% from mode 1 to mode 2 and the standard deviation in the send delay with a 11% increase. The request delay average was the same and the standard deviation decreased 3% whereas the send delay average improved around 1%.

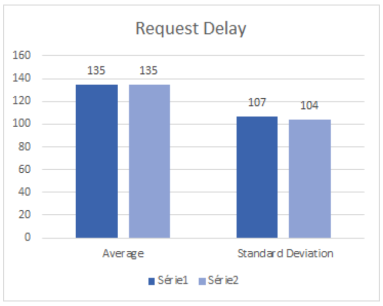


FIGURE 5.4: Request Delay Scenario 1

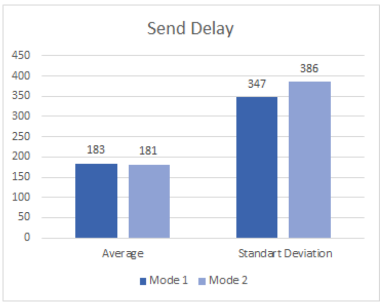


FIGURE 5.5: Send Delay Scenario 1

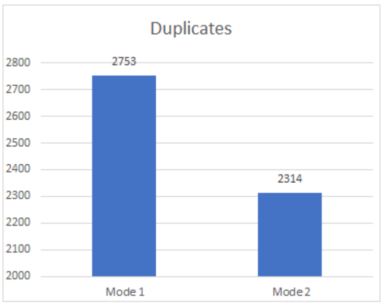


FIGURE 5.6: Duplicates Scenario 1

The second scenario with more VMs keeps the tendency of a lower send delay average with an improve of 3%. The rest of the metrics brought some entropy with the duplicates getting 52% worse from the first mode to the second. The request delay average increased 16% and the standard deviation 31%. The send delay standard deviation improved 4%. These results may be due to some inconsistency in the network since such results like the increased overhead introduced by the increased amount of duplicates is not expected. Also, the increase delay on the requests is not expected but can be justified by the random aspect of Bitswap which still has the possibility of choosing one of the worst peers.

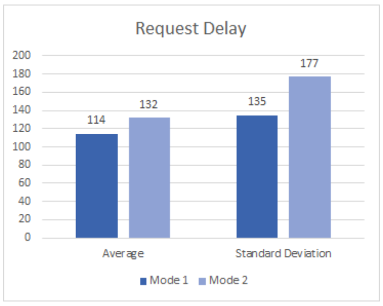


FIGURE 5.7: Request Delay Scenario 2

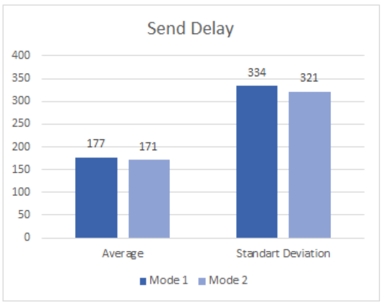


FIGURE 5.8: Send Delay Scenario 2

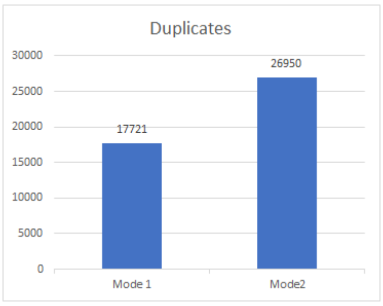


FIGURE 5.9: Duplicates Scenario 2

The results from the first and second scenarios show mild improvements on the send delay metric but proving inconclusive about the request delay and duplicate blocks metrics without a clear trend in both tests. Therefore, it is difficult to extrapolate the outcome of a broader scenario with more nodes or even to a real-world usage of IPFS.

In the third scenario depicted in figures 5.9/5.10/5.11 the request delay average show a minor increase of 2% and a decrease of 4% in its standard deviation between mode 1 and mode 2 of Bitswap. The send delay increased in both average and standard deviation with a 4% and 5% increase respectively. The duplicates comparatively with scenario 2 only show a 7% increase.

This last scenario tried to emulate a real-world utilization of IPFS where multiple users had the same file and multiple users where seeking that same file. However, the results do not show a clear advantage for the proposed change in Bitswap´s way of choosing the next peer to serve a block.

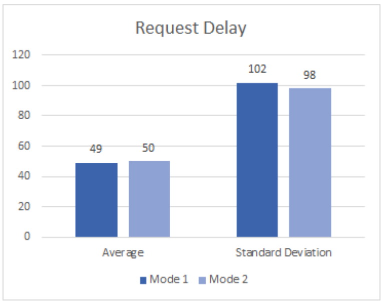


FIGURE 5.10: Request Delay Scenario 3

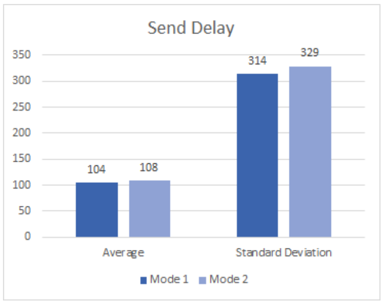


FIGURE 5.11: Send Delay Scenario 3

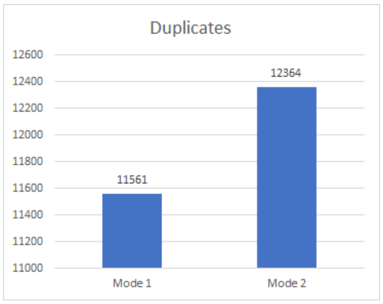


FIGURE 5.12: Duplicates Scenario 3

With such small margins, apart from the second scenario, the reduced sample size of VMs did not lead to a clear conclusion for a real-world scenario with millions of users which is the aim of this alteration. In general, more tests for each scenario would be required to establish a clear trend in all the metrics and subsequently make it easier to extrapolate to real-world scenarios because these two modes of Bitswap operation have a random element to it with regard to how the next peer is selected to provide a block.

One limitation we had during these tests was the Google Cloud overall cost and the time it took to export the block exchange logs (the more time it took the longer the VMs had to stay on and therefore higher costs per test). while the individual cost per VM is relatively cheap, when scaling the experiments it quickly becomes prohibitively expensive

About Bitswap modes 3 and 4, as stated above, after a few tests it was clear that these two modes were hugely affecting IPFS performance, as suggested by us in section 4.2.1. This was probably due to the extra comparisons and loops, and the overloading of a few set of nodes (the closer ones to each peer) not taking full advantage of the concurrency implemented in IPFS. As a consequence of this probably bad coding, we decided not to go all the way with testing these two scenarios that brought a massive disadvantage to the IPFS performance.

5.3 Cost reduction

The integration of P2P at the edge of a CDN infrastructure can help a video streaming company reduce its operational costs.

By redistributing its traffic from their edge servers to nodes that have a specific file cached on the P2P group, video streaming providers can juggle their traffic, following what was proposed on Bitswap operation modes 3 and 4, to maintain a desired QoE, yet still reduce its operational costs.

For example, when no peer has the desired block the node would request them from the edge servers, but if its peers are able to provide the wanted blocks they will be prioritized as providers as long as they can maintain the session latency threshold ensuring a good QoE.

This methodology would allow the company to save as much as the video popularity. With more popularity more the video would be propagated and less bandwidth from the edge server would be required.

It is also an option to deploy additional nodes on the network with the content that a company wants to replicate, in strategic points, to increase resource availability and avoid fetching from edge servers. For this specific option we can estimate the costs reduction by comparing Akamai 1TB of bandwidth ($350) and the monthly cost of one simple Google Cloud VM ($10-$40). This nodes would be controlled by the video streaming company and used to replicate content in order to reduce requests from the CDN infrastructure. In the case of a smaller company, the comparison can be made between the cost of having a CDN versus having a small number of Google Cloud VMs taking part in the P2P mesh.

Additionally, each company could develop incentives to their users for them to participate in the P2P group and even go a step forward and use this incentives so that they can do some proactive caching in their users.

Chapter 6

Conclusion

In order to fully determine the true impact of these improvements and whether or not they give IPFS a performance advantage for video streaming, further extensive testing would be required.

Regarding what was proposed for Bitswap modes 3 and 4, which did end up affecting IPFS performance instead of enhancing it, a solution could be, instead of exhausting the node with the least latency with requests, create a reduced list of peers with lower latencies and select the next block providers from there. These two modes could represent the integration of a P2P mesh with standard CDN infrastructure where peers would exchange blocks amongst themselves but if the average session latency reaches a threshold, blocks would be requested from the edge server in order to maintain QoE.

Aside from that, IPFS is a state of the art P2P file sharing system that provides scalability and performing better with a higher amount of users, perfectly capable of being integrated with a standard CDN infrastructure whilst providing good QoE for its users.

Looking back, and knowing the potential of IPFS, the best we could have made would have been to test the direct integration of IPFS with a CDN like structure and measure, in percentage, how much bandwidth could be saved with the propagation of blocks amongst network users.

With video streaming at an all-time high and continuing to increase, big companies like Youtube, Netflix and Amazon could greatly decrease their server bandwidth usage and therefore saving a lot of money by integrating a P2P group at the edge of their infrastructure.

For further work, it would be interesting to study the particular impact of using a hybrid CDN-P2P on the saving of server bandwidth, and the financial advantages to video streaming companies alongside with incentives to their users towards participating in the P2P group and sharing their cached files.

Appendix A

Github Repositories

Go IPFS - https://github.com/Matias-Correia/go-ipfs

Go Bitswap - https://github.com/Matias-Correia/go-bitswap

Go IPFS Config - https://github.com/Matias-Correia/go-ipfs-config

Test Server - https://github.com/Matias-Correia/go-test_server

Test Management Scripts - https://github.com/Matias-Correia/test-management-scripts

Bibliography

[1] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014. [Cited on pages vii, ix, 2, 7, 18, and 21.]

[2] L. Ceci. (2022) Hours of video uploaded to youtube every minute as of february 2020. [Online]. Available: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/ [Cited on page 1.]

[3] U. Cisco, "Cisco annual internet report (2018–2023) white paper," *Cisco: San Jose, CA, USA*, 2020. [Cited on page 1.]

[4] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," *Journal of Network and Computer Applications*, vol. 62, pp. 18–40, 2016. [Cited on page 1.]

[5] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015. [Cited on page 1.]

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017. [Cited on page 1.]

[7] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*. IEEE, 2015, pp. 73–78. [Cited on page 1.]

[8] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1818–1831, 2017. [Cited on page 1.]

[9] D. F. de Almeida, J. Yen, and M. Aibin, “Content delivery networks-q-learning approach for optimization of the network cost and the cache hit ratio,” in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2020, pp. 1–5. [Cited on page 5.]

[10] E. Baccour, A. Erbad, K. Bilal, A. Mohamed, and M. Guizani, “Pccp: Proactive video chunks caching and processing in edge networks,” *Future Generation Computer Systems*, vol. 105, pp. 44–60, 2020. [Cited on pages 5 and 6.]

[11] M. A. Kader, E. Bastug, M. Bennis, E. Zeydan, A. Karatepe, A. S. Er, and M. Debbah, “Leveraging big data analytics for cache-enabled wireless networks,” in *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–6. [Cited on pages 5 and 6.]

[12] Z. Chen, N. Pappas, and M. Kountouris, “Probabilistic caching in wireless d2d networks: Cache hit optimal versus throughput optimal,” *IEEE Communications Letters*, vol. 21, no. 3, pp. 584–587, 2016. [Cited on page 6.]

[13] B. Cheng, X. Liu, Z. Zhang, H. Jin, L. Stein, and X. Liao, “Evaluation and optimization of a peer-to-peer video-on-demand system,” *Journal of Systems Architecture*, vol. 54, no. 7, pp. 651–663, 2008. [Cited on page 6.]

[14] S. Hasan, S. Gorinsky, C. Dovrolis, and R. K. Sitaraman, “Trade-offs in optimizing the cache deployments of cdns,” in *IEEE INFOCOM 2014-IEEE conference on computer communications*. IEEE, 2014, pp. 460–468. [Cited on page 6.]

[15] R. Martins, M. E. Correia, L. Antunes, and F. Silva, “Iris: Secure reliable live-streaming with opportunistic mobile edge cloud offloading,” *Future Generation Computer Systems*, vol. 101, pp. 272–292, 2019. [Cited on pages 7 and 11.]

[16] M. J. Freedman, E. Freudenthal, and D. Mazieres, “Democratizing content publication with coral.” in *NSDI*, vol. 4, 2004, pp. 18–18. [Cited on pages 7 and 18.]

[17] R. C. Merkle, “A digital signature based on a conventional encryption function,” in *Conference on the theory and application of cryptographic techniques*. Springer, 1987, pp. 369–378. [Cited on page 7.]

[18] R. Wang, J. Zhang, S. Song, and K. B. Letaief, “Mobility-aware caching in d2d networks,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 5001–5015, 2017. [Cited on page 7.]

[19] H. Nam, K.-H. Kim, J. Y. Kim, and H. Schulzrinne, "Towards qoe-aware video streaming using sdn," in *2014 IEEE global communications conference*. IEEE, 2014, pp. 1317–1322. [Cited on page 8.]

[20] F. Qin, Z. Zhao, and H. Zhang, "Optimizing routing and server selection in intelligent sdn-based cdn," in *2016 8th International Conference on Wireless Communications & Signal Processing (WCSP)*. IEEE, 2016, pp. 1–5. [Cited on page 8.]

[21] A. J. Fahs and G. Pierre, "Proximity-aware traffic routing in distributed fog computing platforms," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019, pp. 478–487. [Cited on pages 8 and 9.]

[22] ——, "Tail-latency-aware fog application replica placement," in *International Conference on Service-Oriented Computing*. Springer, 2020, pp. 508–524. [Cited on page 9.]

[23] A. J. Fahs, G. Pierre, and E. Elmroth, "Voilà: Tail-latency-aware fog application replicas autoscaler," in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2020, pp. 1–8. [Cited on pages 8 and 9.]

[24] HashiCorp. Serf: Decentralized cluster membership, failure detection, and orchestration. [Online]. Available: https://www.serf.io/ [Cited on page 8.]

[25] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 15–26, 2004. [Cited on page 8.]

[26] A. Fahs, "Proximity-aware replicas management in geo-distributed fog computing platforms," Ph.D. dissertation, Université de Rennes 1, 2020. [Cited on page 9.]

[27] S. Nacakli and A. M. Tekalp, "Controlling p2p-cdn live streaming services at sdn-enabled multi-access edge datacenters," *IEEE Transactions on Multimedia*, vol. 23, pp. 3805–3816, 2020. [Cited on page 10.]

[28] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *Proceedings of the 17th ACM international conference on Multimedia*, 2009, pp. 25–34. [Cited on page 10.]

[29] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, “Data security and privacy-preserving in edge computing paradigm: Survey and open issues,” *IEEE access*, vol. 6, pp. 18 209–18 237, 2018. [Cited on page 11.]

[30] R. Guo, J. Chen, B. Liu, J. Zhang, C. Zhang, H. Duan, T. Wan, J. Jiang, S. Hao, and Y. Jia, “Abusing cdns for fun and profit: Security issues in cdns’ origin validation,” in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2018, pp. 1–10. [Cited on page 11.]

[31] J. Qu, L. Zhou, G. Zhang, D. Wu, J. Zheng, and Y. Cai, “Secure caching in d2d content sharing,” in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2018, pp. 1–6. [Cited on page 11.]

[32] A. A. Zewail and A. Yener, “Device-to-device secure coded caching,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1513–1524, 2019. [Cited on page 11.]

[33] M. Haus, M. Waqas, A. Y. Ding, Y. Li, S. Tarkoma, and J. Ott, “Security and privacy in device-to-device (d2d) communication: A review,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 1054–1079, 2017. [Cited on page 12.]

[34] F. Zhang, W. He, X. Liu, and P. G. Bridges, “Inferring users’ online activities through traffic analysis,” in *Proceedings of the fourth ACM conference on Wireless network security*, 2011, pp. 59–70. [Cited on page 12.]

[35] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten, “Cookies that give you away: The surveillance implications of web tracking,” in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 289–299. [Cited on page 12.]

[36] P. Koshy, D. Koshy, and P. McDaniel, “An analysis of anonymity in bitcoin using p2p network traffic,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 469–485. [Cited on page 12.]

[37] A. Singh *et al.*, “Eclipse attacks on overlay networks: Threats and defenses,” in *In IEEE INFOCOM*. Citeseer, 2006. [Cited on page 12.]

[38] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on {Bitcoin's}{peer-to-peer} network," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 129–144. [Cited on page 12.]

[39] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260. [Cited on page 12.]

[40] I. Baumgart and S. Mies, "S/kademlia: A practicable approach towards secure key-based routing," in *2007 International conference on parallel and distributed systems*. IEEE, 2007, pp. 1–8. [Cited on page 13.]

[41] A. Salomaa, "Public-key cryptography," 1996. [Cited on page 13.]

[42] E. Jalalpour, M. Ghaznavi, D. Migault, S. Preda, M. Pourzandi, and R. Boutaba, "A security orchestration system for cdn edge servers," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 46–54. [Cited on page 13.]

[43] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229. [Cited on page 13.]

[44] S. Wang, J. Zhou, J. K. Liu, J. Yu, J. Chen, and W. Xie, "An efficient file hierarchy attribute-based encryption scheme in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1265–1277, 2016. [Cited on page 13.]

[45] K. Liang, M. H. Au, J. K. Liu, W. Susilo, D. S. Wong, G. Yang, Y. Yu, and A. Yang, "A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing," *Future Generation Computer Systems*, vol. 52, pp. 95–108, 2015. [Cited on page 13.]

[46] M. R. Baharon, Q. Shi, and D. Llewellyn-Jones, "A new lightweight homomorphic encryption scheme for mobile cloud computing," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 2015, pp. 618–625. [Cited on page 13.]

[47] A. Mtibaa, K. Harras, and H. Alnuweiri, "Friend or foe? detecting and isolating malicious nodes in mobile edge computing platforms," in *2015 IEEE 7th International*

Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2015, pp. 42–49. [Cited on page 13.]

[48] R. Kothari, B. Jakheliya, and V. Sawant, "A distributed peer-to-peer storage network," in *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*. IEEE, 2019, pp. 576–582. [Cited on page 19.]

[49] J. Rosenberg, "Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols," Tech. Rep., 2010. [Cited on page 20.]

[50] K. Thulasiraman and M. Swamy, "5.7 acyclic directed graphs," *Graphs: theory and algorithms*, vol. 118, 1992. [Cited on page 23.]

[51] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970. [Cited on page 34.]

[52] N. D. Mickulicz, U. Drolia, P. Narasimhan, and R. Gandhi, "Zephyr: First-Person Wireless Analytics from High-Density In-Stadium Deployments," in *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2016, pp. 1–10. [Cited on page 35.]