Fog-Based CDN Framework for Minimizing Latency of Web Services Using Fog-Based HTTP Browser

Ahmed H. Ibrahim 1,∗, Zaki T. Fayed 2 and Hossam M. Faheem 3

Citation: Ibrahim, A.H.; Fayed, Z.T.; Faheem, H.M. Fog-Based CDN Framework for Minimizing Latency of Web Services Using Fog-Based HTTP Browser. Future Internet 2021, 13, 320. https://doi.org/10.3390/fi13120320

Abstract: Cloud computing has been a dominant computing paradigm for many years. It provides applications with computing, storage, and networking capabilities. Furthermore, it enhances the scalability and quality of service (QoS) of applications and offers the better utilization of resources. Recently, these advantages of cloud computing have deteriorated in quality. Cloud services have been affected in terms of latency and QoS due to the high streams of data produced by many Internet of Things (IoT) devices, smart machines, and other computing devices joining the network, which in turn affects network capabilities. Content delivery networks (CDNs) previously provided a partial solution for content retrieval, availability, and resource download time. CDNs rely on the geographic distribution of cloud servers to provide better content reachability. CDNs are perceived as a network layer near cloud data centers. Recently, CDNs began to perceive the same degradation of QoS due to the same factors. Fog computing fills the gap between cloud services and consumers by bringing cloud capabilities close to end devices. Fog computing is perceived as another network layer near end devices. The adoption of the CDN model in fog computing is a promising approach to providing better QoS and latency for cloud services. Therefore, a fog-based CDN framework capable of reducing the load time of web services was proposed in this paper. To evaluate our proposed framework and provide a complete set of tools for its use, a fog-based browser was developed. We showed that our proposed fog-based CDN framework improved the load time of web pages compared to the results attained through the use of the traditional CDN. Different experiments were conducted with a simple network topology against six websites with different content sizes along with a different number of fog nodes at different network distances. The results of these experiments show that with a fog-based CDN framework offloading autonomy, latency can be reduced by 85% and enhance the user experience of websites.

Keywords: content delivery networks; content offloading framework; fog-based HTTP browser; fog computing

1. Introduction

The rapid growth of the services provided via cloud computing has significantly affected the capabilities of existing networks, mainly in terms of service latency and QoS. Additionally, due to the COVID-19 pandemic, online services have increased tremendously along with education moving online, working and sharing knowledge becoming remote, and more IoT devices becoming connected daily. Many latency-sensitive services have been affected due to the congestion of the networks with all the requests and data traveling between network nodes. According to the International Data Corporation (IDC), there will be 41.6 billion devices connected to the Internet by 2025, generating 79.4 zettabytes of data [1]. With these issues in mind, content availability is becoming a real challenge.
Content delivery networks (CDNs) have become a crucial component in today’s internet infrastructure to overcome issues regarding content availability. CDNs replicate data to trusted servers located closer to consumers. Cloudflare, one of the main CDN providers in the cloud service industry, managed to reduce the requests to the main cloud server by 65% and saved 60% of the network bandwidth as their cache servers were located near their consumers [2]. Pushing data closer to consumers is a characteristic that is shared between CDNs and the promising fog computing paradigm. According to the OpenFog Consortium, fog computing is defined as a system-level horizontal architecture that distributes resources and services of computing, storage, control, and networking anywhere along the continuum from cloud to things, thereby accelerating the velocity of decision making [3]. In simple terms, fog computing is cloud computing close to consumers.

CDNs improve the latency of centralized cloud servers by pushing content (both static and dynamic) to other cloud servers that are geographically distributed and closer to consumer regions. This approach previously proved its efficiency in reducing network congestion to the main servers and in improving the response time of services; but this approach also moved latency issues to edge servers. Due to the recent increase in the number of online services, these edge servers will have to handle latency issues and network congestion in a different manner. Given that the fog stratum is dynamic, and the fog nodes are not visible nor controlled by service developers and administrators, it becomes hard for cloud providers to utilize the fog stratum efficiently.

This paper proposes a fog-CDN framework capable of reducing the loading time of web services and their latencies by utilizing fog node capabilities. The proposed fog-CDN framework utilizes the headers of the normal HTTP requests to achieve efficient content offloading to the fog nodes. The contents that can be offloaded to the fog nodes are controlled by service engineers, which enable them to control the privacy and ownership of content. We built a new fog-based browser that makes use of the proposed fog-CDN framework.

The rest of the paper is organized as follows: related work is presented in Section 2; Section 3 describes the proposed fog-CDN framework along with its architecture and components; experimental results and implementation details are discussed in Section 4; and, finally, the work is summarized in Section 5, along with a discussion regarding future work and open challenges.

2. Related Work

The fog computing (FC) paradigm is promising in its ability to solve cloud-centric problems, especially in terms of latency and quality of service (QoS). Decreasing the latency of different cloud services is considered one of the main advantages of the FC paradigm. This advantage is inherited from the fog architecture (FA) itself. Various previous research efforts aimed to utilize FA to achieve optimal latency reduction. Most of these studies concerned moving computations to closer nodes in the fog stratum, referred to as computation offloading. In [4], a container-based framework was proposed to deploy microservices in the FA with minimal communication and computation costs. An application offloading strategy in the fog cloud architecture that used the accelerated particle swarm optimization technique was introduced in [5]. A collaborative offloading policy for fog nodes that attempted to minimize the IoT application delays was proposed and evaluated against a developed analytical model in [6]. A request offloading framework was proposed in [7], which employed a collaboration strategy among fog nodes to compute data in a shared mode. Yousefpour et al. [8] provided a comprehensive survey regarding all related research efforts in fog and edge computing. They classified these research efforts into nine categories: foundation, frameworks and programming models, design and planning, resource management, operation, software and tools, testbeds, security, and hardware and protocol stack. Our research is concerned about resource management and operations contributions. These contributions have failed to provide a general-purpose framework that uses FA in a tolerant, maintainable, and controllable manner. Most of them aimed to utilize the processing power available in the fog nodes by offloading the computations closer to the
consumer. While this approach might move in the right direction in order to make use of these computation capabilities in the fog stratum, it still does not provide the standardization and the usability the industry needs. This is mainly due to the number of variations needed for every computation domain, and this affects the offloading strategies. Given that the number of static contents requested via networks represents a considerable percentage of each HTTP request (JS files, images, style sheets, plain texts, videos, etc.), caching near the consumer is another approach to achieve reductions in latency for the consumer.

Content delivery networks (CDNs) are defined by the Internet and Engineering Task Force (IETF) RFC 3466 [9] as a type of network in which servers and contents are arranged in an effective manner with respect to clients. In CDNs, cache servers, also called surrogate servers, which have a copy of the main service contents, are geographically distributed around the world. This enables cloud providers to fulfill end user requests in a faster manner by relying on the nearest cache server. Figure 1 represents an abstract architecture of CDNs where cloud data centers have their own server instances in other geographic locations. The consumer represented as IoT device, PC, or laptop gets their request satisfied from the nearest servers.

Expansion of the network architecture in Figure 1 to show the fog stratum is shown in Figure 2. As shown, the fog stratum can be perceived as a multi-tier network architecture. Between the end consumer and the cloud services, the requests travel through multiple different networks in which these networks have computing and storage capabilities. The devices with computing and storage capabilities are called fog nodes (FNs).
Figure 2. Fog network architecture.

Utilizing FNs in a CDN model is a promising method of reducing latencies in web services. In [10], a caching mechanism (Semi-Edge) was proposed, in which they used in-network caching to achieve latency reduction. The Information Centric Networking (ICN) approach was proposed in [11] to achieve a fog-based CDN. The authors in this contribution also implemented the proposed approach and evaluated the performance in [12]. These contributions rely on the ICN protocol and tackle the caching mechanisms from the perspective of the network layer. These approaches, when applied, restrict the service owner from controlling and monitoring their own data. Using the ICN protocol for adaptive caching was proposed in [13]. The browser caching of the resources also does not provide the prospective latency reduction. This is mainly due to two factors: massive content volumes and storage limitations. An architecture that supports virtual reality content delivery called SRFog is proposed in [14]. This architecture utilizes the Kubernetes-based model on both worker and master nodes. Kubernetes (https://kubernetes.io/ accessed on 29 November 2021) is an orchestration platform that is used for server deployment and scalability as well as management of containerized applications. Using Kubernetes in worker nodes may be feasible for the VR use case, but it is hard to utilize for more general purpose use cases due to the management effort needed to be done on each fog node.
Peer to Peer (P2P) caching is another approach to tackling the cloud CDN limitations. In [15], a hybrid architecture of fog-supported CDN based on P2P is proposed. This approach does not enable service owners to control their data over the edge network. It also does not count for the fog nodes’ storage or bandwidth. Using a cloud hierarchical cloud cache architecture equipped with replications is proposed in [16] to reduce cloud service response time. This approach does not take into consideration the fog nodes’ availabilities, content offloading mechanisms, or registering new services in the fog CDN.

In this paper, we propose a fog-based CDN framework that extends the cloud CDN into the fog stratum. The proposed fog-based CDN framework is built on the application layer, which comes with some advantages, including: cloud–fog transparency, straightforward deployments and updates, data control, easy installation, and fog node monitoring.

3. Proposed Fog-Based CDN Framework

Figure 3 shows the proposed fog-based CDN framework architecture and components. Detailed descriptions and functions of each component are given in the following sections. The architecture diagram depicts one geographic region for abstraction purposes.

Figure 3. Fog-based CDN architecture and components.

3.1. Fog-Based CDN Framework Components

Our proposed framework consists of five components: Fog Registry (FR), Fog Service Locator (FSL), event channel, Fog Worker (FW), and Fog Browser. The FR, FSL, and event channel are deployed in the cloud layer. In the fog layer, the FW is installed on fog nodes to facilitate communication between the cloud and the end devices. Fog Browser is a tool installed on edge devices to take advantage of the proposed framework autonomy.
3.1.1. Fog Registry (FR)

The Fog Registry (FR) is a service that stores all fog-node-related data, as well as fog services available for offloading. Communication between the FR and other components takes place using the RESTful API. The fog nodes register themselves once they boot up. Any web service can integrate with the fog-based CDN framework by registering itself into the FR as well. The FR also keeps track of active event channels that are geographically distributed. Based on the fog node’s geographic location, the FR instructs the nodes to subscribe to specific event channels. Furthermore, the FR publishes the registered services for the registered fog nodes once they get registered or updated.

Since the Fog Registry is keeping track of the registered fog nodes, it can also be extended to compute other fog node related aspects, such as: importance of the fog node and its reliability to other nodes [17,18].

3.1.2. Fog Service Locator (FSL)

The Fog Service Locator (FSL) is a web server extension installed in the cloud environment for web services. The main function of the FSL is to decide whether the service requested is registered in the fog-based CDN or not and whether it is offloaded to a nearby fog node. The FSL intercepts the HTTP request headers coming to the cloud server. If the fog headers exist, the FSL asynchronously checks whether a nearby fog node to the requester can provide quick access for the requested service resources or not. The FSL plays an important role in enabling the fog-based CDN to perform seamlessly for the cloud provider.

3.1.3. Event Channel

The cloud providers need to provide at least one communication channel for all the fog nodes. This communication channel is built using the pub/sub messaging protocol, where different nodes can subscribe to services and the cloud providers can publish updates on specific topics. The cloud providers can provide more event channels for load balancing purposes or more latency optimization if they are geographically distributed.

3.1.4. Fog Worker (FW)

The Fog Worker (FW) is the process that works on the fog node computing processes. The FW is responsible for registering itself in the FR, joining nearby fog nodes and edge device networks, getting offloaded resources for different services, and acting as a CDN node in the fog stratum.

3.1.5. Fog Browser

The Fog Browser is a web browser installed on edge devices such as PCs, laptops, routers, IoT devices, or any programmable device. The main functionality of the Fog Browser is to enable the end device to discover nearby fog nodes and seamlessly retrieve the available resources from those fog nodes. The Fog Browser can work jointly with any existing browser as an extension or plugin. It can be perceived as an underlying layer for the browser. With low-specification edge devices, command line HTTP browsers such as cURL or wget can be extended with fog browser functionalities.

3.2. Fog-Based CDN Framework Workflow

In our proposed framework, the FW is the component that manages the fog and edge strata. It is the coordinator between end consumers and the fog network. The FSL component is the resource inquirer, which decides whether requests can be fulfilled from available fog nodes or not. The FR is the knowledge base that stores data regarding available fog nodes in the network and the available services. The event channel is the communication channel between the FR and the fog nodes. Figure 4 shows the high-level architecture of the proposed framework.
3.2.1. Offloading and Serving as a CDN Node Using Fog Worker (FW)

We present the proposed framework workflow with the Fog Worker (FW) as the first component. The FW has four main functions: self-registration to the FR, joining the local network nodes, pulling the resources, and serving the offloaded resources. The FW is software installed on fog nodes. Here, fog nodes are referred to as hosts. Once the FW is turned on, it sends a registration request to the FR. The registration request contains the following data about the fog node: node key, memory size, host operating system, longitude, and latitude. The self-registration function starts with a registration request sent by a post request following the Restful API standards made available by the FR. The FR then responds back with the registration details and with a universal unique identifier (UUID). The UUID can then be used by the FW to update any host information afterwards. If the FN is already registered, the FR will detect this via the node key and return the previously registered information. After a successful registration, the FN announces its availability in the network for other nodes and edge devices. This is done by a broadcast message in the same local network. After this step, edge devices, more specifically the Fog Browser, know that there is a fog node available in the same network. The FW then subscribes to the event channels. When any new web service is deployed in the cloud servers with fog-CDN enabled, it is published in the event channels and the FW starts pulling all the resources in parallel. If the FW successfully pulls all the available resources, it stores them locally and makes them accessible to the local network via the same cloud URL prefixed with the fog node host IP. For example, if a resource URL such as https://example-service.com/images/fog.png (14 December 2021) is used, it will be made accessible in the local network via http://192.168.1.6/example-service.com/images/fog.png (14 December 2021). Furthermore, the FW reports to the FR that it can serve these resources. Accessing the resources from a local network in this manner provides direct quick access to the resource for all the other devices in the same network. The FW has a microservice running in the background to provide these resources in HTTP. Providing the resources via HTTP, the same protocol they get requested from the cloud, makes the Fog Browser implementation much simpler, as it would only prefix the requested resource with the FN IP. Figure 5 explains all the FW functions in a sequence diagram and the communications with other framework components. The FW and the Fog Browser are both software installed on the fog nodes and edge devices, respectively. This why the host device is outlined by a blue rectangle.
3.2.2. The Fog Registry (FR) Workflow

As shown in the abstract architecture of the proposed framework in Figure 4, the FR is the knowledge base of the proposed framework. The FR stores the fog nodes connected to the cloud providers and the services deployed in the cloud provider’s servers. Each fog node in the outer layer of the network spectrum registers itself to the FR by the FW operating on them. Web service engineers who want to utilize the fog architecture to provide better resource availability need to register their service in the FR with each service deployment. This can easily be part of the web service Continuous Delivery (CI/CD) pipelines. During the deployment stage, the web service (WS) registers itself to the FR. The WS registration requires the following data: service name, resources endpoint, HTTP method, and response schema. Service name is a unique identifier for the service, and it should be unique for all the services hosted by the same cloud provider. The domain name is the preferred choice. The resources endpoint is the URL used by the FW to list all the available resources for this service. This URL should be accessed by the FW using the same HTTP method stored in the FR. The resource listing format by this endpoint can vary from one service to another; therefore, the input schema is provided as an optional field. When it is not provided, the FW will assume the listing to be a JSON list of URLs. Upon the successful registration of a service, the FR publishes information regarding the new service to event channels. After that, all subscribed Fog Workers start the offloading process. The web service registration steps are shown in Figure 6. The FR is unaware of a fog node being active or not. It simply acts as a knowledge base that maps the fog nodes with the offloaded services. The Fog Browser is the component responsible for detecting whether a fog node is active or not.

Figure 5. Functions of the Fog Worker in a sequence diagram.
3.2.3. The Fog Browser Workflow and the Fog Service Locator (FSL) Role

The Fog Browser enables edge devices to utilize the proposed framework and communicate with nearby fog nodes (FNs). It has three main roles: discovering the nearby fog nodes, normal browsing, and retrieving the resources from the nearby fog nodes. First, the Fog Browser sends a broadcast message in the network asking if there are any available fog nodes reachable in the local network. If the FW is operating with any fog node nearby, it will respond back with the UUID they received during the FR registration. To keep the Fog Browser updated, it keeps listening for any new fog node that may join the network. The Fog Browser keeps updating the list of UUIDs of the nearby fog nodes. In any browsing request, the Fog Browser adds two headers to the request: X-FOG-ENABLED and X-FOG-NODES. The fog-enabled header will have a Boolean value that indicates whether the browser has any nearby fog nodes registered or not. If true, the fog nodes header will have a comma separated list of UUIDs of the fog nodes. The Fog Service Locator (FSL) is a layer between web servers and web services. When a request is received with a fog-enabled header set to true, the FSL queries the FR asynchronously with the received UUIDs and service name. After the request gets processed, it adds a response header X-FOG-NODES-QUERY with UUIDs that have the service resources offloaded. The FSL acts as a proxy in case the fog-based CDN headers do not exist or the fog-enabled header is set to false. Figure 7 shows a flowchart of the FSL processes. After that, the Fog Browser receives the response, and, based on the fog node query header, it pulls the requests from the nearby fog nodes. The fog node might get turned off while the request is processing. Here, the Fog Browser uses the original resources as a fallback.
4. Implementation and Experimental Results

Our experiment was conducted in two main phases. In the first phase, our proposed framework components were deployed and installed on different network strata. This ensured the completeness and integrity of the framework since all different components were communicating through the proposed flow seamlessly. In the second phase, we studied the latency improvement of different websites after manually offloading their real static resources in the fog nodes. In this phase, we measured the latency improvements from the Fog Browser component. The experiment was conducted on and only considered the static resources hosted on cloud providers’ servers. In our experiment, we relied on the cloud-based CDN used by the studied websites for their static resource. This allowed us to run our experiment and compare our latency improvement against different cloud CDN providers.

4.1. Prototype Implementation

The FW and the FSL were implemented using the Go programming language (Golang) (https://golang.org/ accessed on 15 November 2021). Golang is a fast-compiling language widely used by the industry. Golang provides concurrency by their built-in libraries. The FW has been developed as a command-line tool with three commands that execute the four main functions of the FW. A summary of each command and its corresponding responsibilities is shown Table 1. Implementing the FW as a command-line tool enables straightforward integrations with different fog node platforms and provides
simple installation and update steps. In our prototype, we used the Kong Gateway (https://docs.konghq.com/gateway-oss/ accessed on 15 November 2021), a lightweight API gateway built on Nginx web server, as our main web server. The FSL was developed as a Kong plugin with the Go Plugin Development Kit (PDK).

| Command Line       | Command Syntax                          | Responsibilities                                      |
|--------------------|-----------------------------------------|------------------------------------------------------|
| Join the Network   | $>./fog-node join-network               | - FR registration.                                   |
|                    |                                         | - Joining nearby FNs.                                |
|                    |                                         | - Subscribing to event channels.                      |
|                    |                                         | - Pulling resources when published.                   |
| Acknowledge Joiners| $>./fog-node acknowledge-joiners        | - Replying to the new Fog Browser discovering the network. |
| Serve resources    | $>./fog-node cdn-server                 | - Serving offloaded resources.                       |

Table 1. A summary of the FW different commands.

The FR was developed using the Django framework with Python 3.9. It was architected as a RESTful API that enables CRUD operations for fog nodes and fog services. An API Key authentication scheme was used for its simplicity. NATS was used as an event channel to publish the newly added or updated resources and for the FW to subscribe to these updates. NATS was used in the industry for distributed systems and microservices to communicate together in an event/data streaming manner. The Fog Browser was developed using Python 3.9. The Fog Browser prototype was implemented as a simple web scraping script that fetched the static resources from nearby fog nodes if advised. All the components were packaged in containers using Docker images and automated using Docker Compose tools.

4.2. Experiment Setup and Results

We have conducted three different experimental scenarios to study the improvements achieved by our proposed framework to service latency. First, we conducted an experiment using a single fog node serving the offloaded resources to a single nearby consumer and measured the latency improvements. Second, we compared the latency improvements evaluated in the first experiment while having the fog node on different network a distance away from the consumer. In other words, the fog node was deployed at different hop counts from the consumer. Third, we have deployed four (4) different fog nodes on the same local network and measured the latency of requesting the services while the resources are offloaded on different fog nodes.

4.2.1. Single Fog Node and Single Consumer Experiment

Our first experiment was conducted on a local network with one Raspberry Pi 3 Model B Plus attached to the network as a fog node. The Raspberry PI had the FW tool installed and running. A MacBook Pro with a 2.9 GHz Dual-Core Intel Core i5 processor was attached to the network as an edge device with the Fog Browser ready for sending consumer requests. The FR API and database were deployed on a cloud server located in Amsterdam, the Netherlands, provided by cloud provider DigitalOcean (https://www.digitalocean.com/ accessed on 6 November 2021). On this server, we installed Kong API Gateway with the FSL as a plugin enabled. A NATS (https://docs.nats.io/ accessed on 6 November 2021) server was installed on the same server as well. This full installation was used to ensure the framework’s completeness and integrity.
To evaluate the latency improvement, the experiment was conducted using the same local network setup with Raspberry PI and a MacBook Pro attached to the network. We manually offloaded the static resources of the studied websites to the fog node. We used Google Chrome development tools to export all the resources of each website. These resources were then offloaded to the Raspberry PI node, acting as a fog node. We targeted six websites in our study, chosen based on their Alexa ranking and their resource sizes. Table 2 lists the studied websites along with their resource sizes.

Table 2. List of the websites studied.

| Website URL                  | Resource Size (Approx.) | Website Classification   |
|------------------------------|-------------------------|--------------------------|
| https://www.gamepedia.com/   | ≈7 MB                   | Video Gaming             |
| https://www.foxnews.com/     | ≈21.5 MB                | News                     |
| https://www.espn.com/        | ≈22.1 MB                | Sport News               |
| https://www.bbc.com/         | ≈8.4 MB                 | News                     |
| https://flickr.com/          | ≈28 MB                  | Photo Sharing            |
| https://www.youtube.com/feed/explore | ≈12 MB                  | Video Streaming          |

Website accessed on 14 December 2021.

The latency difference between a fog-based CDN and a cloud-based CDN was used as the metric to evaluate our framework enhancement made by offloading the resources in a fog node. The latency was calculated as the request response time to load the requested content. We conducted our experiments on the assumption that the fog nodes served more than one edge device. Using the Fog Browser, we assessed each of the studied websites three times in different time periods and measured the latency for both fog-based CDN and cloud-based CDN per run. The first measurement was carried out with the fog-enabled feature turned off, and the second measurement was carried out with fog-enabled feature turned on. While the fog was turned off, the Fog Browser retrieved the websites’ resources using the CDN cloud versions, whereas while the fog enabled feature was turned on, the Fog Browser retrieved the websites’ resources using the fog versions offloaded to the nearby fog nodes. As shown in the bar chart in Figure 8, our experiments show that, with our proposed framework offloading autonomy, service latency can be reduced by 85%, and overall website user experience can be enhanced.

In this experiment, we excluded the resource offloading time to the fog nodes. Our rationale behind this exclusion can be summarized in the following points:

(a) Traditional cloud CDNs geographically distributed around the world need to pull the resources of a given service to their servers. This means that there is also an off-loading time for these cloud CDNs. In our experiments, we normalized these values given that the offloading time to a specific fog node is approximately equal to the offloading time to a specific CDN.

(b) Similar to the cloud CDNs, the fog node can serve more than one edge device, making the offloading time cost negligible to the overall enhancement provided.

(c) The offloading process itself is an ongoing process. Whenever a resource is updated, the FW is notified and pulls the updated resources again.
4.2.2. Fog Node and Hop Counts Experiment

Our second experiment was conducted on a local network, with one Raspberry Pi 3 Model B Plus attached to the network as a fog node. A Linux based machine with a 1.8 GHz Intel Core i7 processor was attached to the network as an edge device with the Fog Browser ready for sending consumer requests. The proposed framework had the same deployment as the first experiment.

We had deployed the fog node on different network distances from the edge device. The first run of this experiment, the fog node was one hop away from the edge device. In the second run, the fog node was two hops away from the edge device. This was achieved by attaching the fog node into another subnetwork using another network switch. In the third run, the fog node was three hops away from the consumer by attaching the fog node behind a proxy server in the subnetwork. Using the Fog Browser, we assessed each of the studied websites (Table 2) three times again in different time periods and measured the latency for both fog-based CDN and cloud-based CDN per run. As shown in the bar chart in Figure 9, our experiments show that increasing the number of hops between the consumer and the fog node increases the latency accordingly. Our proposed framework provided the offloading autonomy needed to achieve service latency reduction by 43.7% for having the fog node away from the consumers by three (3) hops. It achieved a latency reduction of 80.7% when the fog node was two (2) hops away from the consumer.

![Fog-Based CDN Latency vs. Cloud-Based CDN Latency](image)

**Figure 8.** Fog-based CDN latency vs. cloud-based CDN latency.
4.2.3. Retrieving Service Resources from Different Fog Nodes Experiment

Our third experiment was conducted on a local network with four (4) fog nodes. The specifications of these four fog nodes were as follows: one Raspberry Pi 4 Model B, three Raspberry Pi 3 Model B. All four nodes were connected to the local network and have the resources of the six websites offloaded to all of them. A Linux based machine with a 1.8 GHz Intel Core i7 processor was attached to the network as an edge device with the Fog Browser ready for sending consumer requests. The proposed framework had the same deployment as the first experiment.

We ran this experiment four times. In each run, we instructed the Fog Browser to retrieve the resources of each website either from one fog node, two fog nodes, three fog nodes, or four fog nodes, concurrently. We calculated and compared the latency of each experiment run. We used the same one hop latency measurement from the second experiment as the latency measurement of using one fog node, since it had the same experimental configuration. As shown in the bar chart in Figure 10, our experiments show that the latency metric was independent of the number fog nodes, providing the same resources. The experiments achieved almost the same latency enhancement as the first experiment. However, having a number of replicas for the resources in the same local network can improve the reliability of the fog-based CDN.
5. Conclusions

In this paper, a fog-based CDN framework was proposed. The framework provides an autonomous technique for offloading content to nearby fog nodes, utilizing the fog nodes as CDN nodes. This will provide the end user with a better QoS and less latency. Our framework consists of five main components: a Fog Registry (FR), Fog Service Locator (FSL), event channel, Fog Worker (FW), and Fog Browser. These components are deployed at different layers of the fog strata. The FR, FSL, and event channel are deployed at the cloud layer. The FW is installed on the fog nodes at the fog layer. The Fog Browser is installed and used at the edge layer. The end user can make use of this framework using the fog browser as an extension, a plugin to an existing browser. One of the key advantages of our proposed framework is that it does not require any special setup to the network. Our proposed framework respects the privacy of the content, as service engineers control the resources that can be offloaded in the service registration process.

Our experiments were conducted using well-known websites providing different types of static content, which were chosen based on their ranking. We had three experimental scenarios to evaluate the latency enhancement achieved by our proposed framework. These experiments showed that our proposed framework can reduce latency by 85% and enhance website user experience. As a topic of future work, we will consider the number of edge devices served by a fog node in the resource offloading process. We will also incorporate mechanisms to compute other fog node aspects in the Fog Registry which can improve content offloading strategies. Furthermore, we will evaluate our enhanced version of the framework against other metrics, such as the energy consumption of the fog nodes and integrate our framework with other fog-based energy efficient frameworks such as the one proposed in [19].

Author Contributions: Conceptualization, A.H.I.; Data curation, Z.T.F.; Investigation, A.H.I.; Methodology, H.M.F.; Project administration, Z.T.F.; Software, A.H.I.; Supervision, H.M.F.; Validation, H.M.F.; Writing—original draft, A.H.I.; Writing—review & editing, Z.T.F. and H.M.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.
References
1. 41.6 Billion IoT Devices will be Generating 79.4 Zettabytes of Data in 2025—Help Net Security. Available online: https://www.helpnetsecurity.com/2019/06/21/connected-iot-devices-forecast/ (accessed on 29 November 2021).
2. Resource Hub | Cloudflare. Available online: https://www.cloudflare.com/resource-hub/ (accessed on 29 November 2021).
3. Openfog Consortium. OpenFog Reference Architecture for Fog Computing Produced. Ref. Archit. 2017, 20817, 1–162.
4. Zhao, X.; Huang, C. Microservice Based Computational Offloading Framework and Cost Efficient Task Scheduling Algorithm in Heterogeneous Fog Cloud Network. IEEE Access 2020, 8, 56680–56694. [CrossRef]
5. Adhikari, M.; Srirama, S.N.; Amgoth, T. Application Offloading Strategy for Hierarchical Fog Environment Through Swarm Optimization. IEEE Internet Things J. 2020, 7, 4317–4328. [CrossRef]
6. Yousefpour, A.; Ishigaki, G.; Gour, R.; Jue, J.P. On Reducing IoT Service Delay via Fog Offloading. IEEE Internet Things J. 2018, 5, 998–1010. [CrossRef]
7. Al-Khafajy, M.; Baker, T.; Waraich, A.; Al-Jumeily, D.; Hussain, A. Iot-fog optimal workload via fog offloading. In Proceedings of the 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, Zurich, Switzerland, 17–20 December 2018; pp. 349–352. [CrossRef]
8. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. J. Syst. Archit. 2019, 98, 289–330. [CrossRef]
9. A Model for Content Internetworking (CDI) RFC 3466. Available online: https://datatracker.ietf.org/doc/html/rfc3466 (accessed on 29 November 2021).
10. Hua, Y.; Guan, L.; Kyriakopoulos, K. Semi-edge: From edge caching to hierarchical caching in network fog. In Proceedings of the EdgeSys 2018—The 1st ACM International Workshop on Edge Systems, Analytics and Networking, Part of MobiSys 2018, Munich, Germany, 10 June 2018; pp. 43–48. [CrossRef]
11. Alghamdi, F.; Barnawi, A.; Mahfoudh, S. Fog-Based CDN Architecture using ICN Approach for Efficient Large-Scale Content Distribution. Available online: https://link.springer.com/content/pdf/10.1007/978-3-030-03405-4_48.pdf (accessed on 29 November 2021).
12. Alghamdi, F.; Mahfoudh, S.; Barnawi, A. A Novel Fog Computing Based Architecture to Improve the Performance in Content Delivery Networks. Wirel. Commun. Mob. Comput. 2019, 2019, 786094. [CrossRef]
13. Nguyen, Q.N.; Lopez, J.; Tsuda, T.; Sato, T.; Nguyen, K.; Arifuzzaman, M.; Safitri, C.; Thanh, N.H. Adaptive Caching for Beneficial Content Distribution in Information-Centric Networking. In Proceedings of the 2020 International Conference on Information Networking, Barcelona, Spain, 7–10 January 2020; Volume 2020, pp. 535–540. [CrossRef]
14. Santos, J.; van der Hooft, J.; Vega, M.T.; Wauters, T.; Volckaert, B.; de Turck, F. SRFog: A Flexible Architecture for Virtual Reality Content Delivery through Fog Computing and Segment Routing. In Proceedings of the 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), Bordeaux, France, 17–21 May 2021.
15. Shojafar, M.; Pooranian, Z.; Nanjwan, P.G.V.; Baccarelli, E. FLAPS: Bandwidth and delay-efficient distributed data searching in Fog-supported P2P content delivery networks. J. Supercomput. 2017, 73, 5239–5260. [CrossRef]
16. Banditwattanawong, T. Temporal Acceleration for Cloud-CDN-Fog-Edge Hierarchy by Leveraging Proximal Object Replicas. 2019. Available online: https://www.researchgate.net/publication/353621899 (accessed on 29 November 2021).
17. Cauteruccio, F.; Cinelli, L.; Fortino, G.; Savaglio, C.; Terracina, G.; Ursino, D.; Virgili, L. An approach to compute the scope of a social object in a Multi-IoT scenario. Pervasive Mob. Comput. 2020, 67, 101223. [CrossRef]
18. Ursino, D.; Virgili, L. Humanizing IoT: Defining the Profile and the Reliability of a Thing in a Multi-IoT Scenario. In Studies in Computational Intelligence; Springer: Berlin/Heidelberg, Germany, 2020; Volume 846, pp. 51–76. [CrossRef]
19. Ammad, M.; Shah, M.A.; Islam, S.U.; Maple, C.; Alalumie, A.A.; Rodrigues, J.P.C.; Mussa, S.; Tariq, U. A Novel Fog-Based Multi-Level Energy-Efficient Framework for IoT-Enabled Smart Environments. IEEE Access 2020, 8, 150010–150026. [CrossRef]