*Article*

# Reducing Complexity of Server Configuration through Public Cloud Storage

**Sihyung Lee**

School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Korea; sihyunglee@knu.ac.kr; Tel.: +82-53-950-7284

**Abstract:** Hosting networking applications typically involves a server publicly accessible over the Internet. However, preparing a server requires excessive time and effort, particularly for non-expert users. This is because users configure multiple elements including the server, host firewalls, and network firewalls, while considering their interactions. To address this problem, we propose a method that uses public cloud storage, such that all messages are communicated through the storage between the server and clients. As the storage is public and accessible over the Internet, users need not consider firewalls and can focus on configuring the server. We implemented the proposed method for web applications and evaluated its performance by accessing applications from 90 hosts in diverse locations. The evaluation showed that the proposed method does not incur extra delays and clients can access the applications as reliably as the current practice of configuring servers. We also recruited 54 participants and examined the time required to configure a server with the proposed method compared to configuration using current practice. This study demonstrated that the proposed method reduces configuration time from 40−60 to nearly 30 min. We believe that the proposed method provides a basis for improving the manageability of server configuration.

**Keywords:** server configuration; network configuration; public cloud storage; networking applications

**Citation:** Lee, S. Reducing Complexity of Server Configuration through Public Cloud Storage. *Electronics* **2021**, *10*, 1277. https://doi.org/10.3390/electronics10111277

Academic Editor: George Angelos Papadopoulos

Received: 12 May 2021
Accepted: 26 May 2021
Published: 27 May 2021

## 1. Introduction

Many networking applications require servers visible over the Internet. For example, web, mail, chatting, and file-sharing services are provided on servers, such that the services can be accessed from diverse locations. Such publicly accessible servers are often hosted on a temporary basis, for a variety of reasons. Software developers set up servers to validate that mobile applications operate correctly during development phases. Hobbyists run web sites to meet and quickly share photos and files. Gamers launch servers to play online games. Teachers of real-time, online classes set up servers to provide a virtual laboratory for students, such that they can perform experiments and competitions during lectures.

In these temporary settings, it is desirable that a server can be set up with the least amount of delay and complexity, so that it can be accessed immediately on demand. However, for many, non-expert users (In this section, a user refers to the person who hosts a networking application and thus needs to configure a server and firewalls), hosting a publicly accessible server takes significant time and effort [1,2]. This is largely because users must configure a combination of devices, i.e., a server, host firewall, and network firewall, and configuring all of them correctly and consistently requires a high level of knowledge and experience [3,4]. Moreover, network firewalls are often not accessible for configuration, when users connect to networks outside their organizations or when they use public Wi-Fi.

Alternatively, users can rent a server managed by a server-hosting company. However, renting a server accompanies payments, which can be a burden as they accumulate over time [5]. In addition, this type of server typically provides configuration environments different from users' local machines (e.g., Linux vs. Windows), therefore, users must learn these environments before they can fully use the service. The servers also allow a template of predefined functions (e.g., particular types and versions of server languages and database programs), which are not always customizable to user preferences nor can they be easily ported to other servers [6].

This work concentrates on reducing the complexity of hosting a server. In particular, we propose a method that requires the configuration of a single device—the user's own machine, such that the user can fully customize it and work in familiar settings without the need to consider remote devices and their heterogeneous environments. The method uses public, unpaid cloud storage and transmits messages and data through this storage (Figure 1). As the storage is free of charge and publicly accessible, no cost is incurred, and the user is not required to configure firewalls.

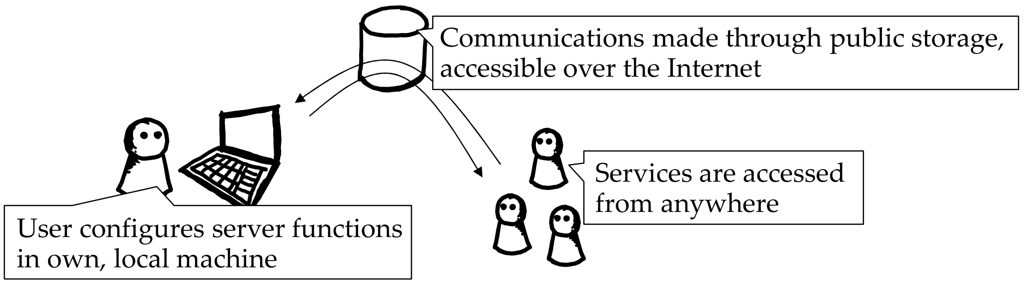

Communications made through public storage, accessible over the Internet
Services are accessed from anywhere
User configures server functions in own, local machine

**Figure 1.** Overview of proposed method.

We implemented the proposed method for web applications, such that web messages and contents are transmitted through public storage. We then made the applications accessible over the Internet and measured its performance when accessed by 90 hosts in geographically diverse locations. We demonstrated that the services can be accessed as quickly and reliably as with the existing methods for hosting servers. We also evaluated the manageability of the proposed method with a user study. A total of 54 participants configured web applications either using the proposed method or the existing methods, and we measured the times required to complete the configurations. The results showed that the proposed method can reduce configuration time from 40−60 to nearly 30 min, and such reduction was greater than 50% for non-expert users. The participants pointed out that the proposed method is simpler, as only a single local device requires configuration.

We summarize our contributions as follows:

- We propose a method that reduces the complexity as well as costs involved in hosting a server for various networking applications. The current practice either requires configuring firewalls [3,4] or paying for renting a server [5,6]. Our proposed method eliminates the need to configure firewalls and incurs no cost, as it uses unpaid storage that is publicly accessible. There exist other studies that relieve the difficulties of server-hosting, and most of them propose user-friendly and centralized configuration interface [1,3,7]. Our work complements these studies; it further relieves the difficulties by reducing the number of devices to configure, as a user needs to configure a single local device but no firewall or remote device. Table 1 compares the different methods.
- We equip the proposed method with techniques that reduce access times, since the method relays messages through public storage and thus may incur delay. We evaluated the techniques by implementing the proposed method for web applications and then by measuring its performance when accessed from 90 hosts that are geographically dispersed. We observed that the applications can be accessed as quickly as with the current practice of hosting servers.
- We performed a user study to demonstrate the effectiveness of the proposed method in reducing the complexity of server-hosting. We recruited 54 participants and asked them to configure web applications either using the proposed method or the existing methods. The study showed that the proposed method reduced configurations

times from 40−60 to nearly 30 min. We also identified reasons why the participants experienced less difficulties with the proposed methods.

**Table 1.** Comparisons among different methods of server-hosting.

| | No Firewall Configuration | No Remote Device Configuration | No Cost for Server Rent | Functions Fully Customizable |
|---|---|---|---|---|
| ① Use own machine for server [3,4] | × | × | ○ | ○ |
| ② Rent server in server-hosting company [5,6] | ○ | × | × | × |
| ③ User-friendly interface [1,7] [1] | △ | △ | △ | △ |
| ④ Proposed method (this paper) | ○ | ○ | ○ | ○ |

[1] ③ User-friendly interface supplements other methods—it can work on top of ①, ②, and ④, thus its property can be either × or ○, depending on which method it operates on.

The remainder of this paper is organized as follows. In Section 2, we present previous work and how this relates to the proposed method. In Section 3, we describe the proposed method in detail. In Section 4, we demonstrate the performance of the proposed method, and subsequently evaluate its manageability in Section 5. Finally, we conclude with an outline of future work in Section 6.

## 2. Related Work

### *2.1. Difficulties in Network Configuration for Non-Expert Users*

Several studies show that non-expert users continue to experience difficulty configuring networking devices and technologies, such as servers, firewalls, and routers. More than 20% of home networking gear is returned, not because the equipment has technical deficiencies, but because users experience difficulty installing and configuring them correctly [2]. Even after successful installation, devices require configuration changes and ongoing maintenance, which often leads to connectivity failures [1] and privacy violations [8]. Troubleshooting tools exist, but are not always user-friendly; and looking up solutions and asking questions on online forums is difficult as many users are unfamiliar with technical terms [9]. The situation becomes aggravated as more wireless devices are connected to home networks; multiple different devices must be configured, their interactions considered, and users must quickly learn complex heterogeneous configuration environments [3].

### *2.2. More Usable and Manageable Network-Configuration Interfaces*

To address the aforementioned difficulties, Jakobi et al. [3] suggest developing visualized systems that non-expert users can better learn and understand. In fact, GUI-based firewall configuration systems have largely replaced command-line-based systems in production networks [10]. These systems can also be centralized, such that users can monitor multiple devices and their interactions in a single screen [7]. Grinter et al. [1] points out that future networking protocols must consider usability and understandability as major design principles, because these principles have been neglected since the inception of the Internet. Overall, previous studies propose user-friendly and centralized networking devices and protocols. Our proposed method further relieves users from complicated configuration tasks; it enables users to configure a single device without considering complex interactions among multiple devices; furthermore, this single device is the user's local machine, so there is no requirement to connect to and configure remote machines, or to work in heterogeneous, unfamiliar environments.

### *2.3. Various Uses of Public Storage*

Public cloud storage is used for various purposes, the most frequent of which is to backup and share data. In particular, the same data are often replicated in multiple storage devices in different locations, so that the data can be quickly accessed from diverse locations; furthermore, failures in one device do not affect the availability of the data. Such replication of data does not incur much delay, as popular storage networks are highly optimized [11,12]. Our proposed method uses this fact to quickly deliver data from a sender to a receiver; a sender writes data to the nearest machine in a storage network, which is quickly synchronized with the machine nearest to the receiver, from which the receiver reads data. Cloud storage can also save space and network bandwidth by informing a client that a file does not have to be uploaded if it has already been received by the same or different clients; this process may leak information about files in storage to arbitrary users, so protection mechanisms can be implemented to prevent such leakage [13]. Our proposed method can also benefit from these improved savings and protection.

The types of data shared in public storage differ. Hu et al. [14] propose to store short video clips in adjacent, free storage, therefore consuming less storage and network bandwidth of online social networks (e.g., Facebook, Twitter, and Instagram). The work by Zhu et al. [15] uses public storage to store sensor data collected from multiple locations, so that these data can be gathered in one place and processed together. In contrast to previous work, our proposed method uses public storage to share data and transmit messages between servers and clients. By doing so, we aim to simplify the configuration of interactive applications that require publicly available servers.

Certain applications can be easily configured in cloud storage without using the proposed method. For example, static web applications can be hosted by dragging and dropping contents into storage and then by sharing the URL [16]. The proposed method is not specific to static web applications but is developed for various networking applications that require servers (more details are shown in Section 3.3); these applications perform a wide range of functions, such as server-side code execution, corresponding access to databases, dynamic content generation, authentication, and user-defined functions in development phase. Configuration of these functions in not always supported in drag-and-drop manner. Furthermore, the drag-and-drop type configuration uses cloud storage to store files, whereas the proposed method uses cloud storage to relay messages between a server and clients. As such, a user does not need to configure firewalls and can equip its own machine with arbitrary server functions.

### *2.4. Relationships with Network Proxies*

The proposed method can be considered to be using public cloud storage as a proxy. A proxy is a machine that acts as an intermediary between a client and a server, such that messages are delivered through the proxy, and it performs useful functions on behalf of the server and client. One common function of proxy is to improve the performance of networking applications; for example, a proxy caches static contents that are recently accessed, so that subsequent accesses can quickly fetch the contents [17]; as another example, a proxy distributes client requests to multiple servers, to balance load across the servers [18]. Proxies are also used to secure a network; for example, a proxy authenticates connections to a server and encrypts/decrypts messages for SSL (Secure Socket Layer) sessions [19]; a proxy may also monitor each session and log important events [20].

Our proposed method has different objectives of using a proxy; it uses public cloud storage to (i) simplify the complexity of server-hosting, since the storage is publicly accessible and does not require firewall configuration; the proposed method also uses unpaid storage to (ii) reduce costs involved in server-hosting. When doing so, the method caches contents to improve performance, similarly to the previous uses of proxies.

## **3. Method of Communication via Public Storage**

We first present an overview of the proposed method in comparison with previous methods (Section 3.1). We then describe details of the method regarding how communication occurs through public storage services (Section 3.2). We also show that the proposed method applies to various applications (Section 3.3).

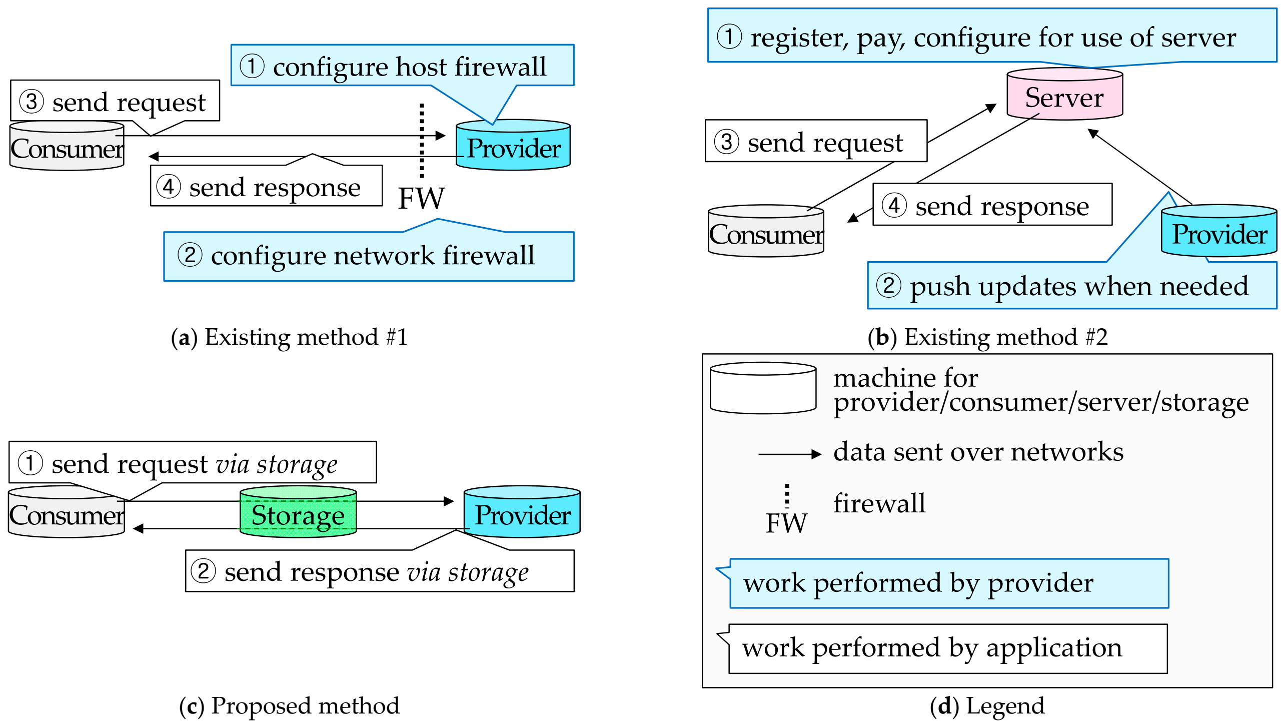
Throughout Sections 3–5, we use the terms in Table 2. A *provider* refers to the person who provides other people with a networking application, and a *consumer* refers to the person who uses such an application. A *server* refers to a publicly accessible server, operated by a server-hosting company. Such a server can be used by a provider to make a networking application accessible to consumers. *Storage* refers to a publicly accessible, unpaid storage facility on the Internet, such as the free space provided by DropBox, Google Drive, and Baidu Cloud. We use the storage facility to replace servers.

**Table 2.** Description of terms.

| Term | Description |
|---|---|
| Provider | Person who hosts a networking application that is accessible by other people |
| Consumer | Person who uses the networking application arranged by a *provider* |
| Server | Publicly accessible server, operated by server-hosting companies |
| Storage | Publicly accessible, unpaid network storage |

### *3.1. Overview of Proposed Method*

We begin by describing two common methods of hosting a networking application visible over the Internet. We then present the proposed method and contrast it with existing methods. Figure 2 illustrates the two existing methods (Figure 2a,b, respectively) along with the proposed method (Figure 2c). A cylinder represents a machine used by a provider, consumer, server, or storage. An arrow represents data communicated between machines. Word balloons show a sequence of operations required for consumers to access the provider's application; in addition to launching the application, the provider must perform the operations in blue balloons; then the rest of the operations in white balloons are automatically performed by the application. Please note that the provider's work in blue balloons is not necessary in the proposed method, as shown in Figure 2c. Each of the three methods in Figure 2a–c are explained in the following paragraphs.



① configure host firewall
③ send request
Consumer
Provider
④ send response
FW
② configure network firewall
(**a**) Existing method #1
① register, pay, configure for use of server
Server
③ send request
④ send response
Consumer
Provider
② push updates when needed
(**b**) Existing method #2
① send request *via storage*
Consumer
Storage
Provider
② send response *via storage*
(**c**) Proposed method
machine for provider/consumer/server/storage
data sent over networks
FW
firewall
work performed by provider
work performed by application
(**d**) Legend

**Figure 2.** Comparison of previous methods with proposed method.

Figure 2a shows the first common method of hosting a networking application. The provider's own machine is used to host the application. To allow customers access to the application, the provider must configure policies at ① the host firewall in the provider's machine and ② the network firewall in the access router. However, configuring a firewall requires knowledge in networking and hence can be time-consuming and error-prone for non-expert users [1–3,8,9]. Moreover, the provider cannot always gain access privileges to network firewalls, e.g., when the provider is working outside the office and does not own the router.

An alternative method is shown in Figure 2b. It uses an external server in a server-hosting company, so the company is responsible for the configuration of firewalls. The provider's tasks are ① to register for the server and then ② to push application contents to the server whenever updates are necessary. However, this often requires a monthly payment; otherwise, functionality can be limited [5]. Even with a payment, the server may support particular applications and options, and the provider is not always given full control and freedom over the server, e.g., a web-server is provided with a set of templates where certain scripts are disabled [6].

To address the shortcomings of the existing methods, we propose a method that uses storage, as shown in Figure 2c. We assume that the provider already has an account for the storage, as free storage services are popular, and many users have accounts for one or more such services [21]. In the proposed method, all messages are sent via *storage*—the sender first writes a message on the storage, which is then read by the receiver. As compared to existing method #1 (Figure 2a), the provider does not need to configure firewalls because the storage is publicly accessible. (Although firewalls do not prevent connection to storage, each directory in the storage, by default, is accessible only by the owner. However, the owner can choose to allow or disallow a particular set of consumers [21]. For example, an arbitrary consumer is allowed to write a request in the directory configured to be writable by every user; then the corresponding response is written to a directory accessible only by the request's sender). Compared to existing method #2 (Figure 2b), no payment is required as the proposed method uses free storage space; furthermore, the provider has full control of functionality because this is chosen and configured on the provider's own machine.

One concern about the proposed method is potential delay—messages are sent via storage, rather than being directly sent between consumer and provider. To reduce the delay, we use the following techniques:

- **Caching.** We cache recently accessed data in storage, so that subsequent access to the same data reads them directly from storage, rather than re-fetching the data from the provider's machine. This process is illustrated in Figure 3. We assume that the storage is initially empty. ① Consumer *C1* requires item *i*, and because *i* does not exist in storage, *C1* requests *i* via storage; ② then *i* is returned via storage; ③ neither *C1* nor the provider delete *i*, therefore, it remains in storage; ④ another consumer *C2* requires the same item *i*, and because *i* is already in storage, *C2* reads it directly from storage, saving time to send a request and receive the item. Cached data can be deleted according to various policies, e.g., remove data that have not been accessed in the past *h* hours.
- **Packaging.** If a set of items are requested together most of the time, we package these items into a single file (e.g., tar or zip) and send it in the first response, therefore removing the need for exchanging multiple pairs of request and response. For example, in web applications, a request for an HTML document often leads to requests for CSS (Cascading Style Sheets), JavaScript, and image files, all of which collectively comprise the same web page; such items can be packaged and sent in a single response. To further reduce delay, we package frequently accessed sets before launching the application and pre-cache them in storage.

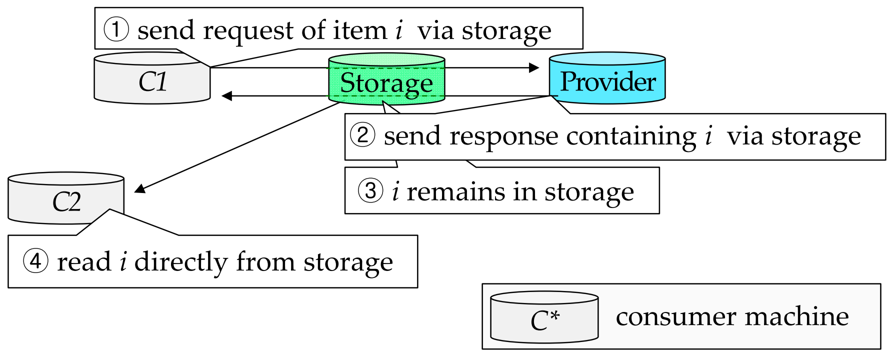

① send request of item *i* via storage
C1
Storage
Provider
② send response containing *i* via storage
③ *i* remains in storage
C2
④ read *i* directly from storage
C*
consumer machine

**Figure 3.** Overview of caching method.

Our evaluation demonstrates that these techniques significantly reduce delay, such that consumers perceive only a minor delay, tolerable in comparison to existing methods (Section 4).

## *3.2. Requesting and Receiving Data via Storage*

We define two types of contents that a networking application provides to consumers: *static* items and *dynamic* items. Using these definitions, we describe the details of the proposed method. *Static* items are data that do not change on user input and thus can be prepared before a consumer requests them; whereas *dynamic* items are those that are made upon request and cannot be prepared in advance. Table 3 presents examples of static and dynamic items in three networking applications. In file-sharing applications, most files are static because file contents do not typically change upon user input. In web and game applications, both static and dynamic items exist. We package and cache static items but not dynamic items, because dynamic items are not predetermined and are generated according to user input upon request.

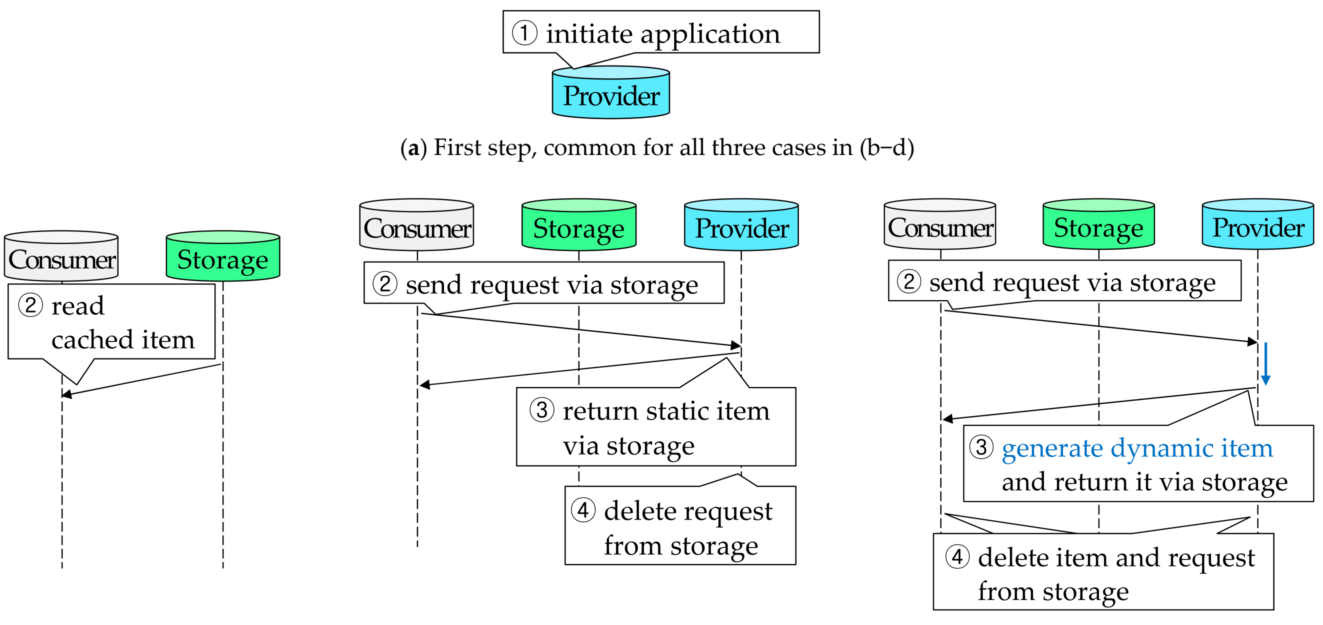
**Table 3.** Example of static and dynamic items in popular networking applications.

| Application | Static Items | Dynamic Items |
|---|---|---|
| File sharing | Most files being shared | |
| Web | CSS, JavaScript, image, and HTML files not generated on user input | User authentication result, contents retrieved by database query |
| Game | Maps, images, audio/video clips | User status updates, chatting messages |

Figure 4 illustrates each step of the proposed method in detail. The first step is shown in Figure 4a, with the following steps then illustrated for each of three different cases, when the desired item is cached in storage (Figure 4b), when a static item is requested (Figure 4c), and when a dynamic item is requested (Figure 4d).

- **Step ①.** The provider initiates the networking application and prepares to serve consumer requests (Figure 4a). Then the following steps ②−④ repeat until the provider terminates the application.
- **Step ②.** A consumer peeks at storage to see whether the desired item is cached (Consumers can learn which files exist in the storage by subscribing to changes in the storage. For example, when we use the DropBox desktop application, we can immediately see updates on a shared DropBox folder, because the application subscribes to changes in the folder [21]). If so, the consumer reads the cached item from storage (Figure 4b). Otherwise, the consumer sends a request to the provider via storage (Figure 4c,d). Please note that multiple consumers can exist and perform these functions simultaneously.

- **Step ③.** The provider processes received requests. If multiple requests exist in the queue, they are processed in order of arrival—the earliest request is processed first. If a request requires a static item, the provider returns it immediately via storage (Figure 4c). If the request requires a dynamic item, the provider generates this item and returns it as soon as possible (Figure 4d). When waiting for dynamic items to be generated, the provider can process the next request.
- **Step ④.** After receiving the requested item, the consumer deletes it from storage if it is dynamic (Figure 4d). Static items remain in storage for future use, i.e., they are cached (Figure 4c) (We target situations where users require a server temporarily, and where this server caches contents that do not exceed the free capacity provided by public cloud storage, i.e., 2−15 GB in general [21]. Therefore, our proposed method does not scale storage space, e.g., as Azure Scaffold does [22]. However, when more caching space is required, cached data can be deleted according to various policies, e.g., remove data that have not been accessed in the past h hours; one can also use cloud storage with larger free space, e.g., 1 TB [23]). The provider also deletes the completed request from storage (Figure 4c,d). The deletions prevent storage from being cluttered by temporary files.


① initiate application
Provider
Consumer
Storage
② read cached item
Consumer
Storage
Provider
② send request via storage
③ return static item via storage
④ delete request from storage
Consumer
Storage
Provider
② send request via storage
③ generate dynamic item and return it via storage
④ delete item and request from storage

(**a**) First step, common for all three cases in (b–d)

(**b**) Desired item cached (**c**) Static item requested (**d**) Dynamic item requested

**Figure 4.** Sequence diagram of proposed method in three different situations.

We now describe rules for writing and reading messages in storage. Many requests and items can be simultaneously written to storage, when multiple consumers access the same provider, and thus it should be possible to differentiate between them.

- **Request.** A request is written to the /request/ folder in storage, which is configured to be writable by consumers allowed to access the application. A request has a file name of the form userID_sequenceNum; userID is the consumer's ID in storage; sequenceNum is the index of the request by the consumer, starting from zero, used to differentiate multiple requests from the same consumer. The request for an item contains this item's ID, and a single request can ask for multiple items by including a list of their IDs. Figure 5 shows an example as follows, steps ②−1 and ②−2 together comprise step ② in Figure 4, and similarly, steps ③−1 and ③−2 comprise step ③ in Figure 4. In step ②−1, consumer #1 writes a request to /request/ folder in storage. This request has a filename consumer1_000, and it asks for file1.html. In step ②−2, the provider reads this request and prepares the requested item.

- **Response.** An item returned by the provider is written to the root folder / in storage if the item can be shared by all consumers; otherwise, it is written to a private /userID/ folder, which is configured to be readable only by the requester. In step ③−1 of Figure 5, the provider writes the requested item file1.html to the /consumer1/ folder, which is then read by consumer #1 as shown in step ③−2.
- **Packaged Response.** In Section 3.1, we mention packaging items to reduce delays. We explain how to package items and how to process packages. Before launching a networking application, the provider identifies static items and then packages those often requested together (We can automate the process of identifying and packaging static items [24], so it will not be a burden for the provider). The file name of a package is itemID_[p]; trailing [p] indicates that the file is a package; itemID is the ID of the item requested first and that leads to the request for other items in the package (e.g., if a request for a.html entails subsequent requests for b.css and c.js, then the file name of the package becomes a.html_[p]). When a consumer requests itemID, the provider returns itemID_[p] if it exists, so that the consumer does not need to send further requests; otherwise, itemID is returned. When the storage space is sufficiently large to accommodate all packages, the provider can choose to pre-cache the packages in storage before launching the application, so that consumers can read desired packages directly from storage without sending requests.
- **Variation in Message Format.** The precise format of requests and responses can vary depending on the networking application. For example, in web applications, requests, and responses can use the HTTP format, therefore using most of the features supported by this protocol. This will also reduce the cost of implementation, because we can reuse most HTTP server and client codes. We describe this implementation in Section 4.

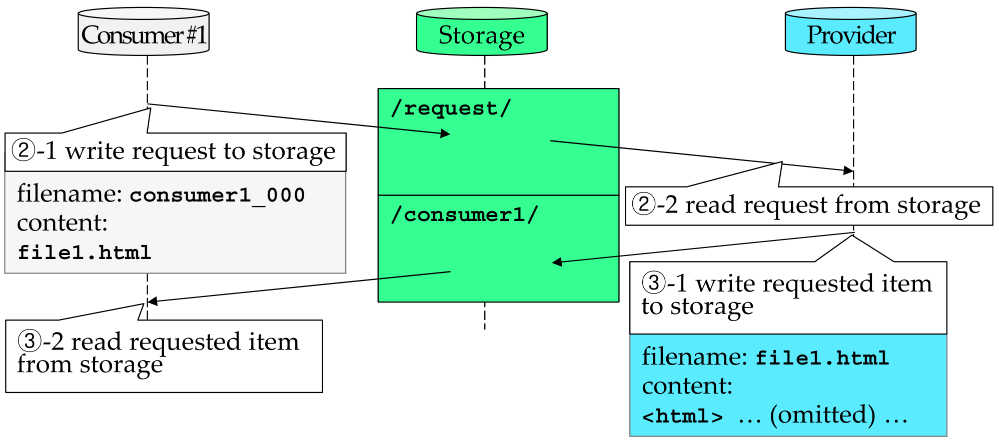


**Figure 5.** Example of writing and reading messages in storage.

## 3.3. *Applications of Proposed Method*

We investigate which networking applications can be hosted by the proposed method. This method works well with most applications that provide static items as these items can be quickly delivered via storage through packaging and caching. Two such popular applications are file sharing (e.g., FTP) and web (e.g., HTTP), as shown in Table 3. In contrast, applications that mainly provide dynamic items may not take full advantage of caching as dynamic items depend on the current input of particular users and are not always reusable. However, we find that dynamic items can be delivered with minimum delay, which is unnoticed or tolerable most of the time (Section 4.2). Such applications include email transfer (e.g., SMTP), text messaging (e.g., AIM), gaming (e.g., Garry's Mod) and Internet telephony (e.g., SIP). In addition to the applications mentioned above, the proposed method can host many custom applications that require publicly accessible servers. These applications include both mobile and desktop applications in various scenarios. For example, during the development phase of a mobile application, the developer can use the proposed method and evaluate networking functions.

In Sections 4 and 5, we evaluated the proposed method for representative applications, i.e., web applications. However, we do not expect strikingly different results with other applications because the simplified steps and performance enhancements apply similarly across different applications.

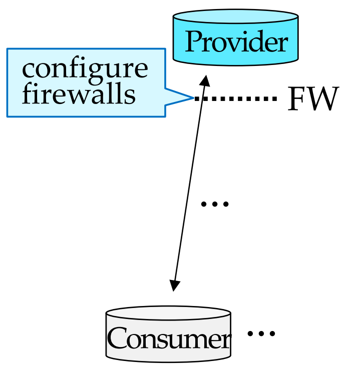
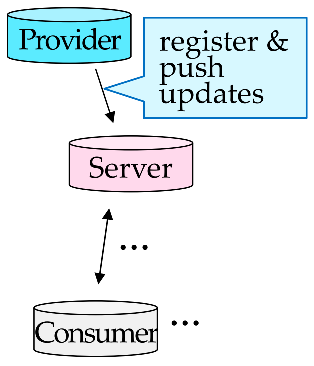
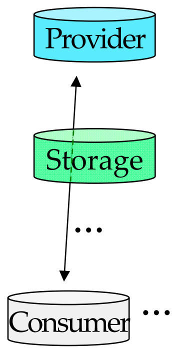
## 4. Performance Evaluation

We implemented the proposed method for web applications and evaluated its performance. We selected web applications, because they comprise a large portion of Internet traffic, nearly 60% [25]. It also delivers a mix of static and dynamic items; hence, we can evaluate the performance of transmitting both. The details of the experiment are explained in Section 4.1 and the results of our evaluation are presented in Section 4.2.

### *4.1. Experimental Setup for Web Applications*

We evaluated three different scenarios, as illustrated in Table 4, and compared their performance. Scenario #1 is when the provider hosts the web-server on its own machine. This corresponds to existing method #1 in Figure 2a, where the provider must configure firewalls in the host and network. Scenario #2 is when the web-server is hosted by a server-hosting company. This corresponds to existing method #2 in Figure 2b, where the provider must register for the server and push updates. Scenario #3 is when storage is used as a communication channel. This corresponds to the proposed method in Figure 2c, where the provider neither configures firewalls nor uses paid services. For each of scenarios #1−3, we measure the time required to access web services from over 90 consumer hosts. The details of each scenario are explained in the following paragraphs.

**Table 4.** Three scenarios, of which we evaluated performance.

| **Scenario #1** | **Scenario #2** | **Scenario #3** |
|---|---|---|
|  |  |  |
| Consumers access web services in provider's machine | Consumers access web services hosted by server-hosting company | Consumers access web services via storage |

For scenario #1, we ran an Apache HTTP server [26] with a MySQL database [27] on the provider's machine, which has a 3.4 GHz CPU (Intel Core i7-6700 (Intel, Santa Clara, CA, USA)) and an 8 GB RAM. We also configured host and network firewalls to allow external connections to the server. For scenario #2, we registered for a web-hosting service [28] with an option of 8 GB memory, 15 GB disk space, and 1 TB of traffic per day. This option was sufficient to perform our experiments without excessive delay. We then pushed web application contents to the server. For scenario #3, we used Google Drive [29] as the storage facility with 15 GB of disk space. The provider machine and its configuration were the same as those used for scenario #1, except that we modified the HTTP server codes using the Google Drive API [30], such that all messages were exchanged via storage, as described in Section 3.2. Before launching the server, we packaged static items that comprise the same web page—these packages remained and were cached in storage after being first requested by a consumer. For all three scenarios, the provider machine was placed in South Korea.

The 90 consumer machines were geographically dispersed: 48 hosts in Asia (China, Korea and Singapore), 30 hosts in North America (Canada and the US), and 12 hosts in Europe (Portugal, Spain, and the UK). These machines were in residential areas and academic networks, and they used Windows 10, 8, and 7. The data rates in these locations vary between 100 Mbps and 1 Gbps. For each of the three scenarios, the same 90 consumer machines accessed the web services for one-week period (from Monday to Sunday) in November 2020, so that the measurement results are not affected by the peculiarities of particular locations, days, and times. During this period, the machines mainly accessed the web services, and did not incur extra heavy network traffic that was not part of the experiment, such as video streaming and large-file downloads. For scenarios #1−2, the consumer machines requested web pages and received them either from the provider machine (for scenario #1) or the server (for scenario #2). For scenario #3, the consumer machines read web pages from storage if they were cached by previous requests; otherwise, the consumer machines requested and received the pages via storage.

We used three web applications in SPECweb benchmark [31], as described in Table 5. These applications ①−③ represent various types of services (e.g., customer-support and online banking sites) and workloads (e.g., large-file downloads and database query), and are designed by analyzing real web applications and server logs. Using the benchmark, we populated the provider machine (for scenarios #1 and #3) and the server (for scenario #2) with web and database contents. The benchmark also provides consumer behavior as a state machine that details the sequence of web pages that consumers access, the probability of accessing such web pages, and the time between consecutive accesses. For example, in ① customer-support application, a consumer may search for a manual, read it, and then consider which page to visit for $t$ seconds; $t$ is determined according to a Geometric distribution with a mean of 60; the consumer then searches for a software patch and downloads it with 30% probability or browses product catalogs with 70% probability. According to these state-machine models, the consumer machines accessed the web applications. Each consumer machine emulated five consumers, because each consumer remains inactive from time to time (e.g., when reading a manual or when considering the next page to visit), so the 90 consumer machines together emulated a total of $90 \times 5 = 450$ consumers.

**Table 5.** Summary of web applications used in the evaluation.

| **Application** | **① Customer-Support** | **② Online Banking** | **③ E-Commerce** |
|---|---|---|---|
| Objective | Search for and download software patches, drivers and manuals; browse product catalogs; search for products | Authenticate, login, and logout users; check transaction details and balance; check, add, and make payments to payees; modify user profile; order checks | Authenticate, login, and logout users; browse and search for computer systems to buy; add products to cart; purchase products in cart |

## 4.2. Measurement Results

Based on the experimental setup, as described in Section 4.1, we measured the time required to access the web applications. In particular, for each instance of access to a web page, we measured the duration from when a page request is made till when the page is fully received and rendered (for the two existing methods in scenarios #1−2), as this is the response time perceived by a consumer [32]. For the proposed method in scenario #3, a consumer machine checks storage for cached items before sending a request, thus we measured the duration from when a consumer machine peeks at storage till when the page is fully received and rendered, as illustrated in Figure 6 (Please note that a web page can consist of multiple items. For example, a request for an HTML document can lead to requests for CSS, JavaScript, and image files, all of which collectively comprise the same web page. In this case, the measured time is from when the first request is sent (for scenarios #1−2) or from when the first peek at storage is made (for scenario #3) till all the items are received and rendered).

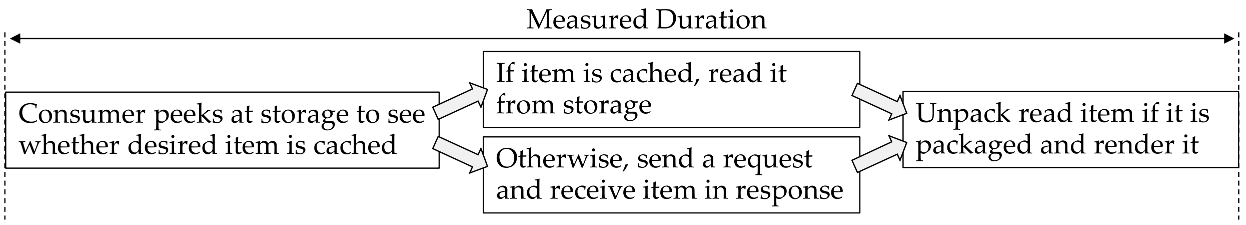
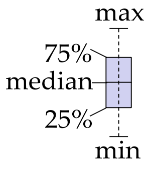
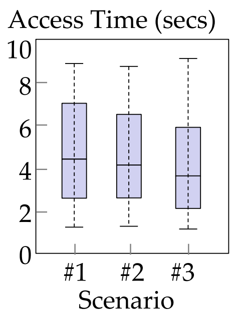
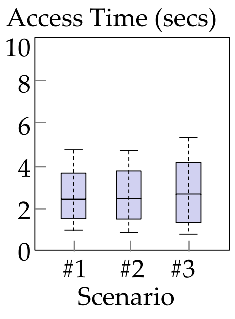
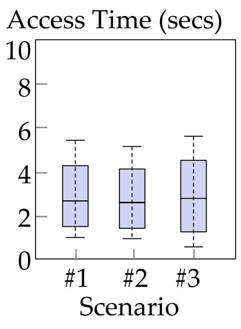

Measured Duration
Consumer peeks at storage to see whether desired item is cached
If item is cached, read it from storage
Otherwise, send a request and receive item in response
Unpack read item if it is packaged and render it

**Figure 6.** Measured time in performance evaluation for scenario #3.

Table 6 summarizes the measurement results. For each of the three web applications ①−③ and each of three scenarios #1−3, we present a box plot to show the distribution of measured times. Among the three applications ①−③, ① customer-support application had the largest access times on average. This is because application ① often transmits larger files (i.e., software patches and drivers of sizes 10−100 MB), whereas applications ② and ③ send smaller pages (<10 MB).

**Table 6.** Distribution of access times for each pair of web applications and measurement scenarios.

| **Application** | **① Customer Support** | **② Online Banking** | **③ E-Commerce** |
|---|---|---|---|
| Access Time Distribution  |  |  |  |

In all three applications, the proposed method (scenario #3) exhibited delays comparable to those of the existing methods (scenarios #1−2), even though the proposed method communicated data via storage (The characteristics presented in this section were observed consistently across the different locations where we performed experiments, i.e., the proposed method had access times comparable to the existing methods). In particular, in application ①, the proposed method was faster than the two existing methods most of the time. These results occurred for the following three reasons. First, once-accessed static items were cached in storage and read directly from storage for subsequent access; this type of caching reduced delays most notably in application ①, because it contains more and larger static items than the other applications. Second, packaging items that are frequently accessed together also contributed to reducing delays; when a consumer requested a page that consists of multiple items, it was not necessary to send multiple requests; only a single request was required to receive the entire page in a package. Lastly, the storage networks could deliver data as quickly as regular Internet connections did; this was also witnessed in previous research [11,12], which shows that popular storage networks are well optimized, such that data are rapidly synchronized. To summarize, the proposed method of communicating via storage did not incur significant performance degradation when compared to existing methods, and outperformed existing methods in applications with many large and static items.

To more accurately analyze the benefits of caching and packaging in the proposed method, we evaluated access times using various subsets of these two techniques. In particular, we further divided scenario #3 into four sub-scenarios: (i) when both caching and packaging are used (scenario #3cp), (ii) when only caching is used (scenario #3c), (iii) when only packaging is used (scenario #3p), and (iv) when neither caching nor packaging is used (scenario #3n). Figure 7 shows the measurement results for application ①; the results were similar for the other two applications. When compared to scenario #3n, where no caching or packaging was used, the access times decreased when either caching or packaging was used (#3c and #3p); when both caching and packaging were used (#3cp), the access times further reduced, beyond those of the two existing methods #1 and #2. This suggests that the two techniques, caching and packaging, complement each other—packaging reduced the number of request-response exchanges, and caching reduced the time for each exchange.

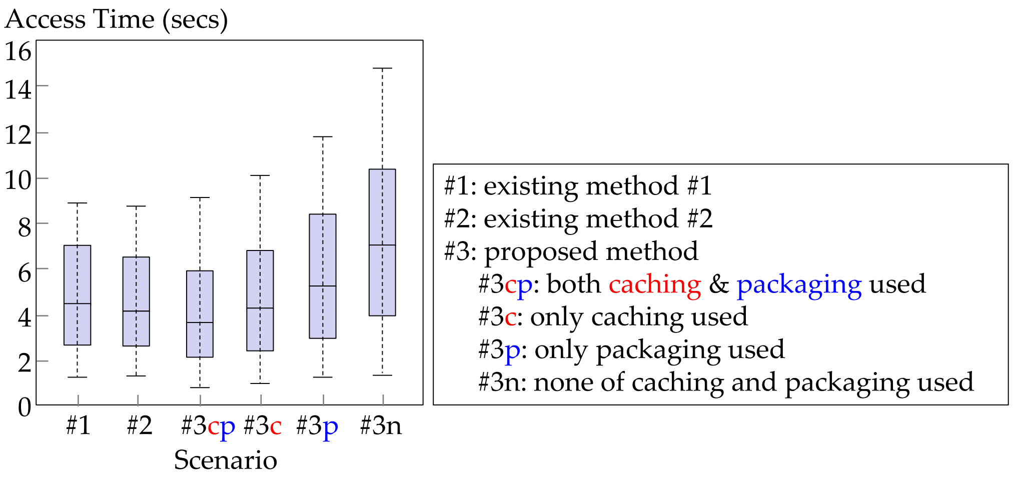


**Figure 7.** Distribution of access times for different combinations of caching and packaging.

## 5. Manageability Analysis

In addition to the measurement of access times in Section 4, we also evaluated the manageability of the proposed method through a user study. The study compared the level of difficulty in configuring and running web applications with the proposed method as opposed to the level of difficulty with the two existing methods. In particular, we measured the time required to complete the configuration of web applications, such that the applications could run and were accessible over the Internet.

We summarize the results as follows:

- We observed decreases in the time required to configure web applications when the proposed method was used, in comparison to the existing methods (e.g., from 40−60 to ~30 min). The decreases were greater for non-expert users (e.g., from 50−90 to ~35 min)
- We identified the main reasons why participants spent more time and made more mistakes when using the existing methods: (i) configuration of multiple, networked elements that interact with one another (e.g., host firewalls and network firewalls) and (ii) configuration and switch between multiple, heterogeneous environments (e.g., local Windows machine and remote Linux machine).

### *5.1. User Study Methodology*

#### 5.1.1. Participants and Group Assignments

We recruited a total of 54 participants for this study, including 11 network administrators and 43 graduate and undergraduate students in technical majors (engineering, science, and mathematics). Among the participants, 11 administrators and 13 students had prior experience configuring firewalls and web servers; in particular, the 11 administrators and two of the 13 students had over two years of experience. The other 30 participants had background knowledge in networks but had not previously configured firewalls; 18 of these had worked with web servers on multiple occasions throughout their academic experience as students; however, the remaining 12 had no such experience. For these 30 participants, we provided appropriate training sessions on firewall and server configurations. Thus, these 30 inexperienced participants assumed the role of relatively new users, whereas the other 24 participants assumed the role of experienced users.

We divided the participants into three groups of equal size (groups #1−3), 18 in each. Groups #1, #2, and #3 configured web applications according to existing method #1, existing method #2, and the proposed method, respectively. We equally distributed experienced participants and inexperienced participants across the three groups so that 8 experienced and 10 inexperienced participants were assigned to each group. From each of the two participant pools, experienced and inexperienced, we randomly assigned participants to one of the three groups.

#### 5.1.2. Task Design

Each participant was given a task to configure web application ① in Table 5. We did not expect strikingly different results with other web applications because the configuration steps are similar across different web applications.

We designed the configuration tasks, such that we can focus on and observe differences among the three comparison groups #1−3, independently of particular web applications. To this end, we assumed that web and database contents are prepared and provided to the participants before the tasks begin, since the process of preparing the contents is the same for the three groups. Given these contents, the participants either configured a local machine and firewalls according to existing method #1 (group #1), configured a remote machine and transferred data according to existing method #2 (group #2), or configured a local machine according to the proposed method (group #3). These steps are summarized in Table 7.

We provided the participants with the contents of web application ①, according to the benchmark [31] we used in Section 4. These includes web contents, their locations in file systems, database contents, and their locations in database systems (i.e., database and table names). The participants then performed the following steps: (i) they copied the contents to the appropriate directories and databases, (ii) they configured servers and databases (i.e., port numbers and home directories) and ran them, and (iii) they configured host and network firewalls, so that the application can be accessed over the Internet. We considered the task complete when the web application became accessible by external consumers. We measured the elapsed time from the initiation of the task to its completion.

The three groups, groups #1−3, followed slightly different steps according to the configuration methods they used. For group #1, the participants configured the same type of server and database in the provider's machine as used for scenario #1 in Section 4—an Apache HTTP server and a MySQL database on a machine with a 3.4 GHz CPU and an 8 GB RAM. The provider's machine used Windows 10 operating system, connected to the Internet through a gateway router (Linksys WRT1900ACS [33]), therefore, the participants configured host firewalls in Windows 10 (inbound policies) and network firewalls in the router (port-forwarding policies). For group #2, the participants registered for a web-hosting service with the same option as used for scenario #2 in Section 4—8 GB memory, 15 GB disk space, and 1 TB of traffic per day. The participants then pushed web application contents to the server with SSH (Secure Shell) and SFTP (Secure File Transfer Protocol). For group #3, the participants used the same type of server and database as in scenario #3 in Section 4—a MySQL database and an HTTP server that communicates via storage.

**Table 7.** User tasks for the three groups.

| Group | Group #1 | Group #2 | Group #3 |
|---|---|---|---|
| Task Summary | Host a web application in a provider's machine | Host a web application in a server in a server-hosting company | Host a web application via storage |
| Task Steps | • configure a web-server and databases (e.g., port number, home directory)<br>• copy contents to proper directories and populate databases<br>• configure host firewalls<br>• configure network firewalls | • register for a server in a server-hosting company<br>• choose options to use in the server to configure it (e.g., server-side web language, database type)<br>• copy contents to proper directories and populate databases | • configure a web-server and databases (e.g., port number, home directory)<br>• copy contents to proper directories and populate databases |

#### 5.1.3. Procedure

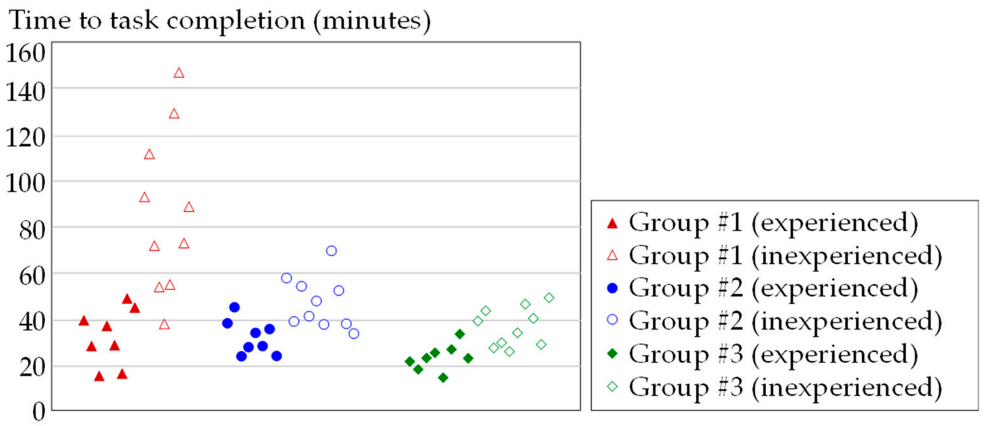
In group #1, for participants without experience with firewall and server configurations, we allocated two hours of training on the operation and configuration of firewalls and web servers. In group #2, for the inexperienced participants, we allocated an hour of training on web-hosting, service registration, selecting options, and using SSH and SFTP to transfer data. In group #3, for the inexperienced participants, we allocated one hour of training on web-server operation and configuration, similarly to group #1. The training used a set of slides and documents based on formal training materials designed for network engineers [34–36]. These materials are provided to the participants after the training, so that they can refer to the materials while performing the tasks.

Participants then began completing the tasks, as described in Section 5.1.2. Upon completion, we invited specific feedback, such as: What caused them to make mistakes? Where they spent most of their time? What aspects of the configuration process they found easy and/or difficult? This immediate feedback was efficient and detailed because the participants' memories about tasks they just completed were fresh.

### *5.2. Results of User Study*

We tested for significant differences in the mean time-to-task-completion (i) between group #1 (those who used existing method #1) and group #3 (those who used the proposed method) and (ii) between group #2 (those who used existing method #2) and group #3. We hypothesized that we would likely observe faster performance in group #3, which confirms that the proposed method requires less time than existing methods #1 and #2. In particular, our null hypotheses $H_0'$s state that the proposed method shows no difference compared to the two existing methods in configuration time, and the alternative hypotheses $H_a$'s state that the proposed method outperforms the existing methods. To test our hypotheses, we conducted the one-sided $t$-test on the mean time-to-task-completion. The test results in a $p$-value, the smallest value of the significance level $\alpha$ for which the null hypothesis can be rejected. The smaller the $p$-value becomes, the more compelling the evidence that the null hypothesis be rejected, and the alternative hypothesis be accepted. A smaller $p$-value favors group #3—the use of the proposed method to configure a web application.

The hypotheses and results of the tests are summarized in Figure 8, including scatter plots of the individual samples. The filled triangles and empty triangles represent samples for group #1, experienced participants and those with no prior configuration experience, respectively. Similarly, the circles represent samples for group #2, and rectangles represent samples for group #3.



$H_0$: $t_1 = t_3$ (group #1 and group #3 bear no significant differences)

$H_a$: $t_1 > t_3$ (proposed method takes less time than existing method #1)

| $t_1$ (μ,σ) | $t_3$ (μ,σ) | *t*-statistic | *p*-value |
|---|---|---|---|
| (62.55, 38.10) | (31.53, 10.38) | 3.33 | 0.001658 |

$H_0$: $t_2 = t_3$ (group #2 and group #3 bear no significant differences)

$H_a$: $t_2 > t_3$ (proposed method takes less time than existing method #2)

| $t_2$ (μ,σ) | $t_3$ (μ,σ) | *t*-statistic | *p*-value |
|---|---|---|---|
| (40.19, 11.76) | (31.53, 10.38) | 2.34 | 0.01261 |

**Figure 8.** Summary of statistical tests for significant differences in mean time-to-task-completion.

All hypothesis tests were in favor of group #3; the *p*-values were less than 0.05, and the null hypotheses showing no difference were therefore rejected at the $\alpha = 0.05$ significance level. The scatter plots also show that group #3 outperformed groups #1 and #2, particularly for inexperienced users. We interpreted the feedback from the participants and identified the reasons as follows.

- **Group #1** (configuration with provider's own machine). The participants found it difficult to understand the interactions among multiple, networked elements (i.e., web-server, host firewall, and network firewall), and often failed to configure all these elements consistently. For example, the port number of an inbound packet admitted by the network firewall should match the number admitted by the host firewall; this number should also match the number that the web-server is running on; only then, the packet can be successfully received by the server. When such configurations contain errors, the participants could not easily figure out where to look at and how to solve the problems.
- **Group #2** (configuration of servers in web-hosting services). The participants spent a significant portion of time populating data into a remote server in the web-hosting service. In particular, they pointed out that the server being remote, as opposed to being local, made the configurations difficult and time-consuming, because they had to transfer data and configure a remote machine through SSH and SFTP. For inexperienced users, it was difficult to grasp these remote access methods. It also took time to learn remote configuration environments (Linux-based) that are different from local environments (Windows-based).
- **Group #3** (proposed method). The participants had difficulty when they did not have prior experience configuring web applications, similarly to those in groups #1 and #2.

However, unlike group #1, the participants were not required to configure multiple firewalls and ensure consistency; in addition, unlike group #2, most of the work was completed locally on the provider's machine, therefore, the participants were not required to switch between remote and heterogenous environments.

### *5.3. Limitations of User Study*

The generality and representativeness of our results can be constrained by the number of participants of the study and by the configuration tasks used in the study. The number of participants was capped at 54 as network administrators and students typically have limited time only. However, these participants represent users with a wide range of experience levels in network and server configurations, varying from greater than 5 years to no experience. We also observed consistent perspectives about the benefits of the proposed system and disadvantages of the existing methods.

The configuration tasks were designed to highlight differences among the three comparison groups #1−3, independently of particular web applications. As such, the configuration steps used in the tasks are similar across different web applications, so we do not expect strikingly different results with other web applications. The participants also stated that the reasons why they spent most of their time were mainly related to the differences among the three configurations methods and not to the particular web applications used in the tasks. However, depending on what applications are configured, the exact configuration times can differ. As we continue the development of the proposed system, we plan to recruit more participants and publish additional results in more diverse settings.

## 6. Conclusions

We propose a method that assists users in setting up a publicly accessible server. In particular, we target situations where users require a server temporarily and where this server has a medium to small hit rate—some hundreds or thousands per hour. The method uses public, unpaid cloud storage to deliver data between the server and clients. Because the storage is already accessible over the Internet, users are not required to configure remote firewalls and can focus on local settings, therefore reducing the complexity involved in server configuration. Using the proposed method, we ran web applications and measured their performance. We found that users can interact with the applications without significant delay when compared with existing methods. We also conducted a user study to compare the manageability of the proposed method with current practice of configuring a server. The study proved that the proposed method can reduce configuration times from 40−60 to nearly 30 min, and such reduction is particularly noticeable for non-expert users.

We plan to apply the proposed method to various other networking applications to evaluate and improve performance and manageability. We also intend to explore additional methods for configuring a combination of networking devices, such that users can easily verify their consistency and correctness.

**Funding:** This research was supported by Kyungpook National University Research Fund, 2019.

**Institutional Review Board Statement:** Ethical review and approval were waived for this study, since it did not collect personal information and asked about the technologies presented in this work. Even so, we explained to the participants about the details of the study and asked for their consent.

**Informed Consent Statement:** Informed consent was obtained from all participants involved in the study.

**Data Availability Statement:** This study utilized the SPECWeb benchmark at http://www.spec.org/web2009/docs/ (accessed on 10 March 2021).

**Acknowledgments:** We thank all the participants of the performance analysis and user study for their time and valuable suggestions.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## References

1. Grinter, R.E.; Edwards, W.K.; Chetty, M.; Poole, E.S.; Sung, J.; Yang, J.; Crabtree, A.; Tolmie, P.; Rodden, T.; Greenhalgh, C.; et al. The ins and outs of home networking: The case for useful and usable domestic networking. *ACM Trans. Comput. Hum. Interact.* **2009**, *16*. [CrossRef]
2. Shehan, E.; Edwards, W.K. Home networking and HCI: What hath god wrought? In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, 28 April–3 May 2007; pp. 547–556. [CrossRef]
3. Jakobi, T.; Ogonowski, C.; Castelli, N.; Stevens, G.; Wulf, V. The catch(es) with smart home: Experiences of a living lab field study. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, Denver, CO, USA, 6–11 May 2017; pp. 1620–1633. [CrossRef]
4. Lee, S.; Kim, H.S. End-user perspectives of Internet connectivity problems. *Elsevier Comput. Netw.* **2012**, *56*, 1710–1722. [CrossRef]
5. Dao Research. *Amazon Web Services Cost Surprises. Oracle White Paper*; DAO: San Francisco, CA, USA, 2019. Available online: https://www.oracle.com/a/ocom/docs/dc/em/dao-research-aws-cost-surprises-white-paper.pdf (accessed on 10 March 2021).
6. Zhang, Z.; Wu, C.; Cheung, D. A survey on cloud interoperability: Taxonomies, standards, and practice. *ACM SIGMETRICS Perform. Eval. Rev.* **2013**, *40*, 13–22. [CrossRef]
7. Hafidh, B.; Osman, H.A.; Arteaga-Falconi, J.S.; Dong, H.; Saddik, A.E. SITE: The simple Internet of Things enabler for smart homes. *IEEE Access* **2017**, *5*, 2034–2049. [CrossRef]
8. Edwards, W.K.; Grinter, R.E.; Mahajan, R.; Wetherall, D. Advancing the state of home networking. *Commun. ACM* **2011**, *54*, 62–71. Available online: https://dl.acm.org/doi/pdf/10.1145/1953122.1953143 (accessed on 10 March 2021). [CrossRef]
9. Dalal, A.C.; Chan, J.; Mitchell, K. A preliminary study of the role of language in home network troubleshooting. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, Glasgow, UK, 4–9 May 2019; pp. 1–6. [CrossRef]
10. CheckPoint. Available online: https://www.checkpoint.com/ (accessed on 10 March 2021).
11. Bocchi, E.; Drago, I.; Mellia, M. Personal cloud storage benchmarks and comparison. *IEEE Trans. Cloud Comput.* **2017**, *5*, 751–764. [CrossRef]
12. Ravenscraft, E. Cloud Storage Showdown: Dropbox vs. Google Drive. Zaiper, December 2019. Available online: https://zapier.com/blog/dropbox-vs-google-drive/ (accessed on 10 March 2021).
13. Pooranian, Z.; Chen, K.; Yu, C.; Conti, M. RARE: Defeating side channels based on data-deduplication in cloud storage. In Proceedings of the IEEE INFOCOM, Honolulu, HI, USA, 15–19 April 2018; pp. 444–449. [CrossRef]
14. Hu, H.; Wen, Y.; Niyato, D. Public cloud storage-assisted mobile social video sharing: A supermodular game approach. *IEEE J. Sel. Areas Commun.* **2017**, *35*, 545–556. [CrossRef]
15. Zhu, C.; Leung, V.; Rodrigues, J.; Shu, L.; Wang, L.; Zhou, H. Social sensor cloud: Framework, greenness, issues, and outlook. *IEEE Netw.* **2018**, *32*, 100–105. [CrossRef]
16. Almeida, B. Google Cloud Website Hosting with Google Cloud Storage. NetApp, February 2020. Available online: https://cloud.netapp.com/blog/google-cloud-website-hosting-on-google-cloud-storage-gcp-cvo-blg (accessed on 10 March 2021).
17. Abrams, M.; Standridge, C.R.; Abdulla, G.; Williams, S.; Fox, E.A. Caching Proxies: Limitations and Potentials. Computer Science Technical Report of Virginia Polytechnic Institute and State University. 1995. Available online: https://eprints.cs.vt.edu/archive/00000427/ (accessed on 10 May 2021).
18. Piorkowski, A.; Kempny, A.; Hajduk, A.; Strzelczyk, J. Load balancing for heterogeneous Web servers. In Proceedings of the International Conference on Computer Networks, Ustron, Poland, 15–19 June 2010; pp. 189–198. Available online: https://link.springer.com/book/10.1007/978-3-642-13861-4 (accessed on 10 May 2021).
19. Abdo, J.B. Authentication proxy as a service. In Proceedings of the International Conference on Fog and Mobile Edge Computing, Valencia, Spain, 8–11 May 2017; pp. 45–49. [CrossRef]
20. Taniguchi, Y.; Tsutsumi, H.; Iguchi, N.; Watanabe, K. Design and evaluation of a proxy-based monitoring system for OpenFlow networks. *Sci. World J.* **2016**, *2016*, 1–10. [CrossRef] [PubMed]
21. Dropbox vs. Google Drive: Comparing Cloud Storage Services. Digital Guide, January 2021. Available online: https://www.ionos.com/digitalguide/server/tools/dropbox-vs-google-drive/ (accessed on 10 March 2021).
22. How to Achieve Your Governance Needs in Azure. Azure Scaffold: The Framework Your Business Needs to Effectively Implement Microsoft Azure. *Cloud Direct Whitepaper.* Available online: https://cdn2.hubspot.net/hubfs/452680/Azure%20Scaffold%20whitepaper.pdf (accessed on 10 May 2021).
23. Jeffrey, W. Baidu Launches Overseas 1TB Free Cloud Storage Dubox, but You Have to Be Careful when Using it. Panda Yoo News, September 2020. Available online: https://pandayoo.com/2020/09/15/baidu-launches-overseas-1tbs-free-network-disk-dubox-but-you-have-to-be-careful-when-using-it/ (accessed on 10 May 2021).
24. Ali, W.; Shamsuddin, S.; Ismail, A. A survey of web caching and prefetching. *Int. J. Adv. Soft Comput. Appl.* **2011**, *3*, 1–27. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1059.1593&rep=rep1&type=pdf (accessed on 10 May 2021).

25. Maier, G.; Feldmann, A.; Paxson, V.; Allman, M. On dominant characteristics of residential broadband Internet traffic. In Proceedings of the ACM Internet Measurement Conference, Chicago, IL, USA, 4–6 November 2009; pp. 90–102. [CrossRef]
26. Apache HTTP Server Project. Available online: https://httpd.apache.org/ (accessed on 10 March 2021).
27. MySQL: The World's Most Popular Open Source Database. Available online: https://www.mysql.com/ (accessed on 10 March 2021).
28. DOTHOME: Server-Hosting Company. Available online: https://www.dothome.co.kr/ (accessed on 10 March 2021).
29. Google Drive. Available online: https://www.google.com/drive/ (accessed on 10 March 2021).
30. Google Drive API. Available online: https://developers.google.com/drive (accessed on 10 March 2021).
31. SPECweb2009 Benchmark. Available online: http://www.spec.org/web2009/docs/ (accessed on 10 March 2021).
32. Asrese, A.S.; Eravuchira, S.J.; Bajpai, V.; Sarolahti, P.; Ott, J. Measuring Web latency and rendering performance: Method, tools, and longitudinal dataset. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 535–549. [CrossRef]
33. Linksys WRT1900ACS Dual-Band Wifi Router. Available online: https://www.linksys.com/us/p/P-WRT1900ACS/ (accessed on 10 March 2021).
34. Stoddard, D.; Thomas, M. *Network Security First-Step: Firewalls*; Cisco Press: Freehold, NJ, USA, 2012.
35. Laurie, B.; Laurie, P. *Apache: The Definitive Guide*; O'Reilly Media: Sebastopol, CA, USA, 2002.
36. Barrett, D.; Silverman, R.; Byrnes, R. *SSH, The Secure Shell: The Definitive Guide*; O'Reilly Media: Sebastopol, CA, USA, 2005.