symmetry

Article

Design and Implementation of Virtual Private Storage Framework Using Internet of Things Local Networks

Hwi-Ho Lee [1], Jung-Hyok Kwon [2,*] and Eui-Jik Kim [1,*]

[1] School of Software, Hallym University, 1 Hallymdaehak-gil, Chuncheon, Gangwon-do 24252, Korea; leehh7028@hallym.ac.kr
[2] Smart Computing Laboratory, Hallym University, 1 Hallymdaehak-gil, Chuncheon, Gangwon-do 24252, Korea
* Correspondence: jhkwon@hallym.ac.kr (J.-H.K.); ejkim32@hallym.ac.kr (E.-J.K.); Tel.: +82-33-248-2333; Fax: +82-33-242-2524

Received: 30 January 2020; Accepted: 18 March 2020; Published: 24 March 2020

Abstract: This paper presents a virtual private storage framework (VPSF) using Internet of Things (IoT) local networks. The VPSF uses the extra storage space of sensor devices in an IoT local network to store users' private data, while guaranteeing expected network lifetime, by partitioning the storage space of a sensor device into data and system volumes and, if necessary, logically integrating the extra data volumes of the multiple sensor devices to virtually build a single storage space. When user data need to be stored, the VPSF gateway divides the original data into several blocks and selects the sensor devices in which the blocks will be stored based on their residual energy. The blocks are transmitted to the selected devices using the modified speedy block-wise transfer (BlockS) option of the constrained application protocol (CoAP), which reduces communication overhead by retransmitting lost blocks without a retransmission request message. To verify the feasibility of the VPSF, an experimental implementation was conducted using the open-source software *libcoap*. The results demonstrate that the VPSF is an energy-efficient solution for virtual private storage because it averages the residual energy amounts for sensor devices within an IoT local network and reduces their communication overhead.

Keywords: constrained application protocol; Internet of Things local network; sensor device; speedy block-wise transfer; virtual private storage

1. Introduction

As the amount of data produced by individuals has increased explosively in recent years, so has the demand for storage solutions for efficiently storing, accessing, and managing user data [1–4]. Two representative storage solutions are typically considered for storing user data: cloud storage and on-premises storage. Cloud storage stores user data on remote servers maintained by third-party service providers such as Google Drive, Apple iCloud, and Dropbox [5–7]. On-premises storage stores user data on local servers dedicated to specific users such as network-attached storage (NAS) [8]. Cloud storage does not require hardware installation at a personal site but is vulnerable to data leakage because it can be accessed by anyone over the Internet [9–11]. Consequently, users are reluctant to store critical data using cloud storage. In contrast, on-premises storage can help to prevent data leakage because the user can set the security policy of the local server to control how the data are stored and who has access [12,13]. However, local storage demands a high cost to install and maintain the hardware, with the risk that all data can become inaccessible due to a local server failure [14].

Meanwhile, with the spread of the Internet of Things (IoT), embedded sensor devices interacting each other over an IoT local network have been deployed in real-world applications [15–18]. As the requirements for IoT applications become more complex, sensor devices tend to be designed with sufficient storage space (e.g., on-board memory and external memory) to perform specific operations reliably [19]. Accordingly, to solve the problems of existing storage solutions, the extra storage space of sensor devices can be merged to build local storage virtually (virtual storage) that does not require additional hardware installation costs and effectively prevents data leakage. Moreover, even when some sensor devices fail, user data stored in the other sensor devices are accessible. Even though the virtual storage is often too small due to the limited number of sensor devices or extra storage size, it can be very useful for storing various kinds of data that require privacy, but are only up to kilobyte in size, such as personal health and financial data. However, sensor devices have physical resources that are inherently limited, such as batteries and microcontroller units (MCUs) [20]. Consequently, the following must be considered when designing virtual storage using such sensor devices.

- Long lifetime: The most important role of storage solutions is to guarantee seamless and reliable data access. However, with virtual storage, some data may not be accessible due to the energy depletion of sensor devices. Each sensor device may have different residual energies—a sensor with low residual energy is discharged before other sensor devices if it is frequently used for storing data. Therefore, before storing data, the virtual storage should be able to select the sensor device based on its residual energy.
- Lightweight data transfer: For storing a large amount of data, the virtual storage must divide the data into multiple blocks and transmit the data to sensor devices in a constrained IoT environment. This causes the sensor devices to suffer from high communication overhead, which leads to a long delay and high energy consumption.

In order to build the storage using sensor devices within an IoT local network, the distributed file system (DFS) solutions such as Ceph, Lustre, Hadoop Distributed File System (HDFS), and Google File System (GFS) can be used [21–24]. The DFS makes it possible to divide a data file into several parts and store them in multiple different devices. To this end, the DFS maintains the list and information of all stored files as metadata (i.e., type, size, attributes, etc.) via name nodes or meta-servers. However, the existing DFS solutions do not consider the limited physical resources of sensor devices composing the IoT local network. Thus, DFS-based storage is highly likely to suffer from unreliable data access due to the energy-depleted sensor devices and the high communication overhead. The recent paradigm of fog computing (a.k.a., edge computing) seems to be an ideal solution to build virtual storage using sensor devices in the IoT local network, because it is expected to benefit an IoT local network by reducing energy consumption, increasing bandwidth utilization, and enhancing security/privacy compared to cloud computing [25–27]. However, when it comes to implementing virtual data storage based on the fog computing paradigm, the research on its implementation technology is still in its infancy. In order to implement virtual data storage based on fog computing, the technologies capable of logically integrating and managing the storage space of edge devices are required, and in addition, various aspects of the data storage system, such as energy consumption, bandwidth, fault tolerance, scalability, and security, should be considered. However, only a few research efforts have been put into fog computing-based virtual data storage [28,29].

This paper presents a virtual private storage framework (VPSF) using Internet of Things (IoT) local networks. The VPSF uses the extra storage space of sensor devices in an IoT local network to store users' private data, while guaranteeing expected network lifetime, by partitioning the storage space of a sensor device into data and system volumes, and, if necessary, logically integrating the extra data volumes of the multiple sensor devices to virtually build a single storage space. When user data need to be stored, the VPSF gateway divides the original data into several blocks and selects the sensor devices in which the blocks will be stored based on their residual energy. The blocks are transmitted to the selected devices using the modified speedy block-wise transfer (BlockS) option of the constrained application protocol (CoAP), which reduces communication overhead by retransmitting lost blocks without a retransmission request message [30–32]. To verify the feasibility of the VPSF, an experimental implementation was conducted using the open-source software *libcoap* [33,34]. The results demonstrate that the VPSF is an energy-efficient solution for virtual private storage because it averages the residual energy amounts for sensor devices within an IoT local network and reduces their communication overhead.

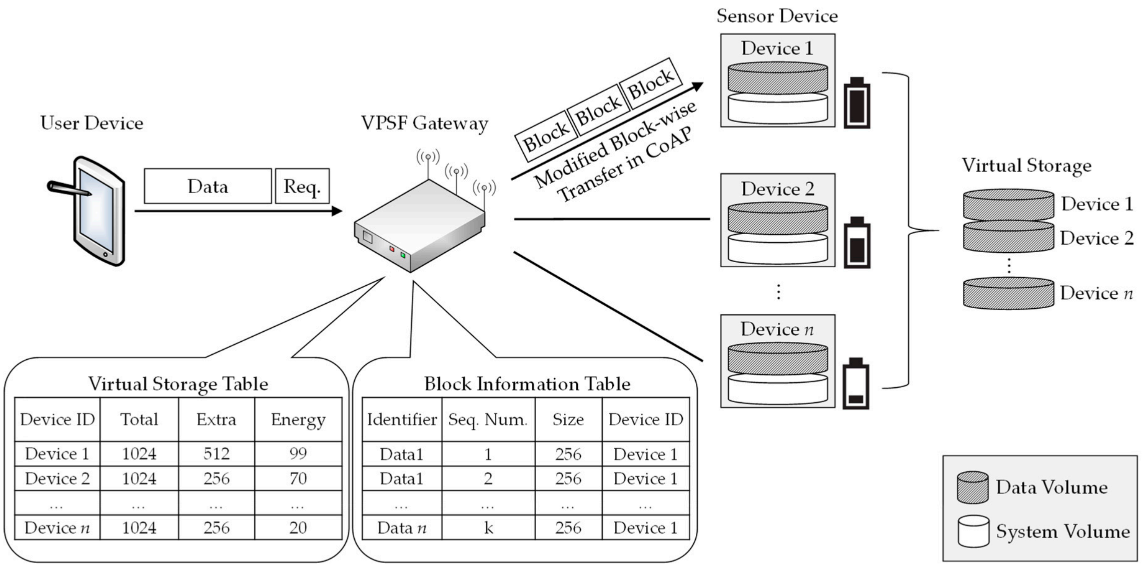
The rest of this paper is organized as follows. Section 2 describes the VPSF design in detail. Section 3 presents the results of the implementation and performance evaluation. Finally, Section 4 concludes the paper.

2. VPSF Design

The VPSF stores user data in the extra storage space of sensor devices while guaranteeing the expected network lifetime of the IoT local network. Accordingly, the VPSF averages the remaining energy of the sensor devices across the IoT local network by building virtual private storage consisting of a specific group of sensor devices, and reduces communication overhead by using the modified BlockS of the CoAP. In this section, the VPSF design is described in detail.

2.1. VPSF Architecture

Figure 1 illustrates the VPSF architecture consisting of the user device, VPSF gateway, and sensor devices. It is assumed that VPSF gateway and sensor devices are static and all sensor devices are within the transmission range of the VPSF gateway. The user device is responsible for the request for data storing and retrieval. In the figure, the user device requests to store the user data. If the user device has the data that need to be stored, it transmits both the storing request message and the data to the VPSF gateway. In contrast, it transmits only the retrieval request message to the VPSF gateway when data retrieval is needed. The storing and retrieval request messages are identified by the VPSF gateway through a message type field in the header of each message.



Virtual Storage Table

Device ID	Total	Extra	Energy
Device 1	1024	512	99
Device 2	1024	256	70
...	...	...	...
Device *n*	1024	256	20

Block Information Table

Identifier	Seq. Num.	Size	Device ID
Data1	1	256	Device 1
Data1	2	256	Device 1
...	...	...	...
Data *n*	k	256	Device 1

Figure 1. Virtual private storage framework (VPSF) architecture.

The VPSF gateway builds virtual storage by logically integrating the extra data volumes of the multiple sensor devices and manages it by maintaining a virtual storage table (VST) containing the total amount of extra data volumes of all sensor devices in the network and the information of each sensor device (device ID, extra data volume size, and residual energy). For storing data, the VPSF gateway divides the user data into several blocks and selects the sensor devices in which the blocks will be stored based on their residual energy. Then, the VPSF gateway forwards the blocks to the selected sensor devices and maintains a block information table (BIT) containing the information of the stored blocks (original user data identifier, block sequence number, block size, and device ID of the sensor device where the block is stored). The selected sensor device stores only the information contained in the payload of the blocks in binary format. For data retrieval, upon receiving the retrieval request message, the VPSF gateway checks the identifier of the requested data and then searches the BIT to find the device ID of sensor devices where the blocks for the requested data are stored. It then requests the blocks of the corresponding sensor devices in order of sequence number. After the VPSF gateway obtains all blocks, it integrates them into a single data set and transmits the integrated data to the user device.

Each sensor device serves as storage for the blocks. The storage space of a sensor device is divided into data and system volumes by the VPSF gateway—each sensor device separately maintains two volumes. The data volume is a part of the sensor device's storage space that is used to build virtual storage, and the system volume is a storage space required to perform the sensor device's unique operations (e.g., sensor and actuator control). Therefore, upon receiving the blocks from the VPSF gateway, the sensor device stores them in the data volume within its storage space. For data retrieval, it transmits the requested blocks to the VPSF gateway.

The VPSF can have a variety of use cases, such as healthcare, financial, and multimedia data storage, in which it is necessary to consider the physical resources of sensor devices running VPSF such as battery capacity and storage space. One of the potential use cases for VPSF is the personal healthcare data storage. In this use case, the wearable or mobile user devices periodically collect users' health information and generate the healthcare data files. The generated files are transmitted to the VPSF gateway. Then, the VPSF gateway divides the files into multiple blocks and store them in the extra storage space of the sensor devices. In the use case, users maintain personal health information via their own local storage, which is logically created by merging the extra storage space of sensor devices. Therefore, VPSF eliminates the need for users to store sensitive health information in storage managed by third-party service providers, and allows users to maintain it by themselves without additional hardware installation.

2.2. *Operations of VPSF Gateway*

The VPSF gateway performs three operations: (1) virtual storage building, (2) sensor device selection, and (3) consecutive block retransmission. In the following subsections, the operations of the VPSF gateway are described in detail.

2.2.1. Virtual Storage Building

In a virtual storage building, the VPSF gateway first partitions the storage space of a sensor device into two volumes: system and data. The storage space of the system volume should be sufficient to store the system data needed for installing and running the operating system (OS), applications, and task data generated by processing specific tasks. It is difficult to determine the storage space for the system volume because each sensor device may use a different OS and applications. Consequently, we consider two assumptions to determine the storage space for the system volume. First, the storage space required to store system data is determined by examining the filled storage space when the sensor device does not perform any task, provided as a constant value in VPSF. Second, the sensor device generates task data of the same size at periodic intervals (task data generation interval) and temporally accumulates the data in the storage space. The sensor device transmits the accumulated task data to the intended destination and removes it from the storage space at specific intervals (task data removal interval). The storage space for the data volume (extra storage space) for the *i*-th sensor device (DV_i) is given by

$$DV_i = T_i - Sys_i - Task_i = T_i - Sys_i - SizeTask_i \frac{RemTask_i}{GenTask_i} \tag{1}$$

where T_i, Sys_i, $Task_i$, $SizeTask_i$, $GenTask_i$, and $RemTask_i$ are the total storage space, the storage space required to store system data, the storage space required to store task data, the task data size, the task data generation interval, and the task data removal interval for the *i*-th sensor device, respectively. The VPSF gateway then logically integrates the data volume of all sensor devices to build virtual private storage. The total storage space of virtual storage is equal to the sum of the storage space of the data volumes. The total storage space of the virtual private storage and the storage space of the data volume for each sensor device are listed in the VST. The VST is updated when a new sensor device is added or the existing sensor device is depleted.

2.2.2. Sensor Device Selection

The purpose of sensor device selection is to guarantee the expected lifetime of the IoT local network by averaging the residual energy of sensor devices, which may differ for each device. Accordingly, when building virtual private storage, the VPSF gateway examines the residual energy of each sensor device and maintains it using the VST. The residual energy values of all sensor devices are represented by $\mathbf{E} = [e_0, e_2, \ldots, e_{n-1}]$, where n is the number of sensor devices. Sensor device selection is initiated when the VPSF gateway receives the storing request message, which contains the size of the user data, from the user device. Algorithm 1 defines the sensor device selection procedure. In the algorithm, the VPSF gateway iteratively performs sensor device selection. Before initiating sensor device selection, the VPSF gateway initializes variables to zero: the number of iterations (k), the device ID of the sensor device selected in the k-th iteration (SSD_k), and the sum of the data volume of the selected sensor devices (SDS). It then calls the list of residual energy for each sensor device from the VST and initiates sensor device selection. During sensor device selection, the VPSF gateway selects a sensor device with the highest residual energy (based on the VST) and verifies whether the data volume of the selected sensor device is large enough to store the user data; if it is not, the sensor device with the next highest residual energy is selected from the remaining list. The VPSF gateway then sums the data volumes of all selected sensor devices and compares the result with the size of the user data. This operation repeats until the sum of the data volumes of the selected sensor devices becomes larger than the size of the user data (*SizeUsr*). After terminating sensor device selection, the VPSF gateway divides the user data into multiple identically-sized blocks and sequentially transmits the blocks to the selected sensor devices.

Algorithm 1. Sensor device selection procedure

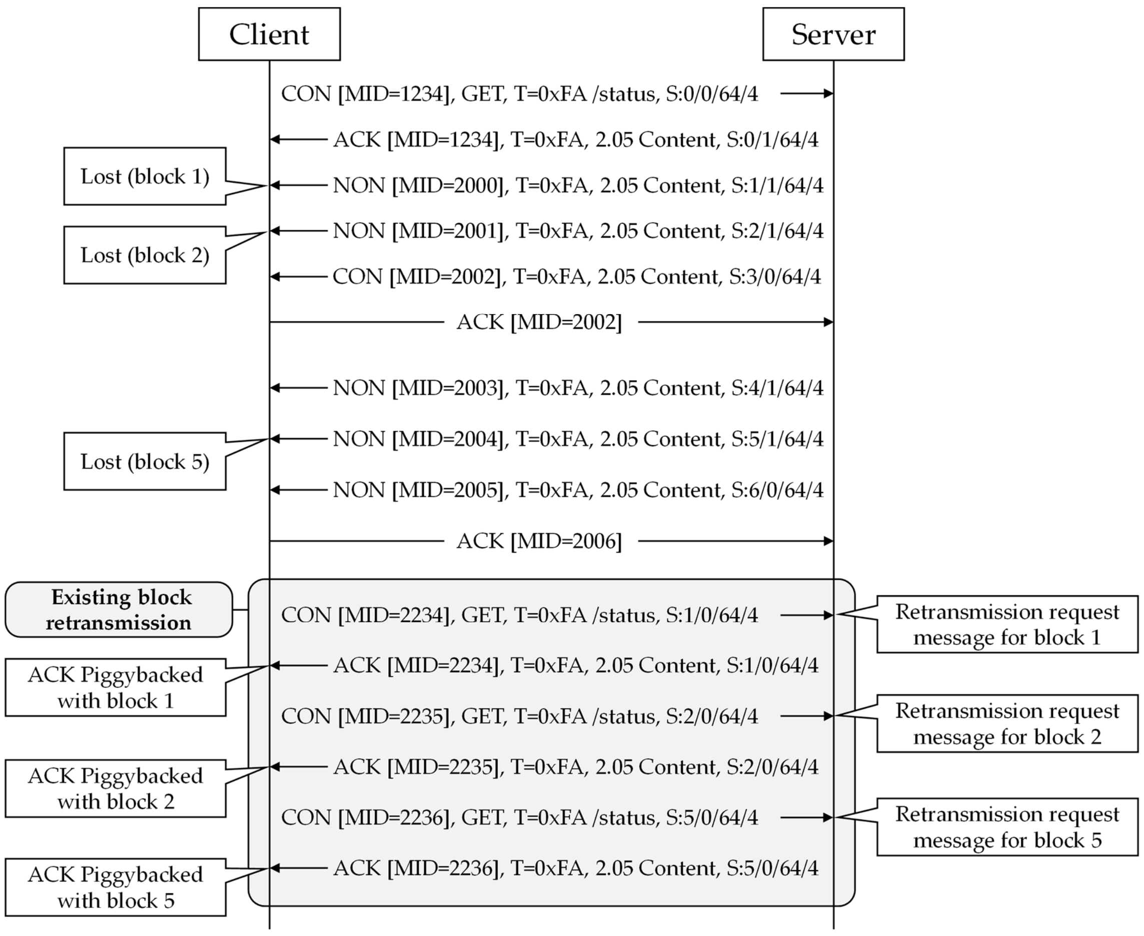
```
1:   INITIALIZE $k$ to 0, $SSD_k$ to 0, $SDS$ to 0   // Initialize variables
2:   Call $\mathbf{E}$ from VST
3:   REPEAT   // Initiate sensor device selection
4:     $maxE \leftarrow \max(\mathbf{E})$   // Find the highest residual energy
5:     FOR each sensor device, $i$, $i \in [0, n-1]$
6:       IF $\mathbf{E}[i] == maxE$   // Find the device ID of the sensor device with the highest residual energy
7:         $SSD_k \leftarrow i$   // Select a sensor device
8:       ENDIF
9:     ENDFOR
10:    $\mathbf{SSD}[k] \leftarrow SSD_k$   // List the selected sensor devices
11:    $\mathbf{E}[SSD_k] \leftarrow 0$   // Remove the residual energy of the selected sensor device from E
12:    $SDS \leftarrow SDS + DV_i$   // Sum the data volumes of the selected sensor devices
13:    $k \leftarrow k + 1$   // Increment the number of iterations by one
14:  UNTIL $SizeUsr \leq SDS$   // Terminate sensor device selection
15:  RETURN SED   // Return the device ID of the selected sensor devices
```

2.2.3. Consecutive Block Retransmission

Consecutive block retransmission aims to reduce communication overhead in the VPSF. Accordingly, the BlockS option of the CoAP is modified to retransmit lost blocks without any request message and then applied to the VPSF. The existing BlockS option of the CoAP uses the non-confirmable (NON) message, which does not require acknowledgment when transmitting the blocks. This allows the server to transmit multiple blocks consecutively. For the existing BlockS option, the confirmable (CON) message that requires acknowledgment is regularly transmitted to verify the client's state. The server transmits the CON message when the number of consecutive transmissions reaches the maximum window size (SPDYWND), which is predefined based on the client's capacity limit. The number of blocks transmitted consecutively is equal to the value of SPDYWND. However, for retransmission, the existing BlockS option only uses the CON message, causing unnecessary retransmission request messages to be transmitted repeatedly.

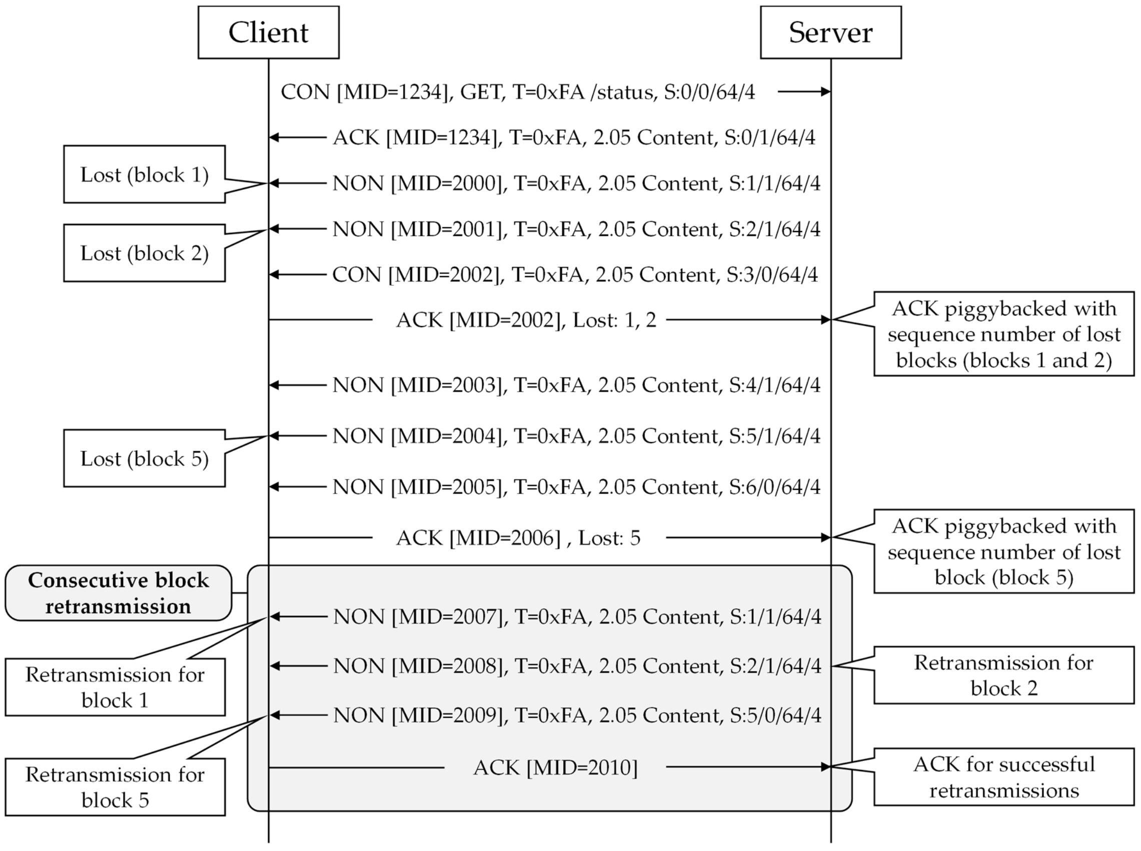
To solve this problem, in VPSF, the existing protocol is modified to inform the server of the sequence number of the lost blocks. Upon receiving the blocks, the client identifies the lost blocks by examining the gap between their sequence numbers and maintains the sequence number of the lost blocks as a list. The client then piggybacks the list to the acknowledgment and transmits it to the server, thereby providing the server with the sequence number of the lost blocks. Consequently, the server consecutively transmits the lost blocks as NON messages without any retransmission request messages.

Figure 2 illustrates an operational example of the existing and modified BlockS option of the CoAP. MID is the message ID, T is the token, and S is the value of the BlockS option (the sequence number of the blocks, whether more blocks are following, the size of the block, and SPDYWND). In the example, the size of the block is set to 64 B, and SPDYWND is set to 4. The example demonstrates that the modified protocol reduces communication overhead by eliminating retransmission request messages.


Client
Server
CON [MID=1234], GET, T=0xFA /status, S:0/0/64/4
ACK [MID=1234], T=0xFA, 2.05 Content, S:0/1/64/4
Lost (block 1)
NON [MID=2000], T=0xFA, 2.05 Content, S:1/1/64/4
Lost (block 2)
NON [MID=2001], T=0xFA, 2.05 Content, S:2/1/64/4
CON [MID=2002], T=0xFA, 2.05 Content, S:3/0/64/4
ACK [MID=2002]
NON [MID=2003], T=0xFA, 2.05 Content, S:4/1/64/4
Lost (block 5)
NON [MID=2004], T=0xFA, 2.05 Content, S:5/1/64/4
NON [MID=2005], T=0xFA, 2.05 Content, S:6/0/64/4
ACK [MID=2006]
Existing block retransmission
CON [MID=2234], GET, T=0xFA /status, S:1/0/64/4
Retransmission request message for block 1
ACK Piggybacked with block 1
ACK [MID=2234], T=0xFA, 2.05 Content, S:1/0/64/4
CON [MID=2235], GET, T=0xFA /status, S:2/0/64/4
Retransmission request message for block 2
ACK Piggybacked with block 2
ACK [MID=2235], T=0xFA, 2.05 Content, S:2/0/64/4
CON [MID=2236], GET, T=0xFA /status, S:5/0/64/4
Retransmission request message for block 5
ACK Piggybacked with block 5
ACK [MID=2236], T=0xFA, 2.05 Content, S:5/0/64/4

(**a**)

Figure 2. *Cont.*


Client
Server
CON [MID=1234], GET, T=0xFA /status, S:0/0/64/4
ACK [MID=1234], T=0xFA, 2.05 Content, S:0/1/64/4
Lost (block 1)
NON [MID=2000], T=0xFA, 2.05 Content, S:1/1/64/4
Lost (block 2)
NON [MID=2001], T=0xFA, 2.05 Content, S:2/1/64/4
CON [MID=2002], T=0xFA, 2.05 Content, S:3/0/64/4
ACK [MID=2002], Lost: 1, 2
ACK piggybacked with sequence number of lost blocks (blocks 1 and 2)
NON [MID=2003], T=0xFA, 2.05 Content, S:4/1/64/4
Lost (block 5)
NON [MID=2004], T=0xFA, 2.05 Content, S:5/1/64/4
NON [MID=2005], T=0xFA, 2.05 Content, S:6/0/64/4
ACK [MID=2006] , Lost: 5
ACK piggybacked with sequence number of lost block (block 5)
Consecutive block retransmission
NON [MID=2007], T=0xFA, 2.05 Content, S:1/1/64/4
Retransmission for block 1
NON [MID=2008], T=0xFA, 2.05 Content, S:2/1/64/4
Retransmission for block 2
NON [MID=2009], T=0xFA, 2.05 Content, S:5/0/64/4
Retransmission for block 5
ACK [MID=2010]
ACK for successful retransmissions

(**b**)

Figure 2. Comparison of two operational examples: (**a**) existing speedy block-wise transfer in CoAP; (**b**) modified speedy block-wise transfer in CoAP.

3. Implementation and Performance Evaluation

The VPSF performance was evaluated through implementation and extensive experiments. The user device was implemented using a personal computer running the Windows 10 OS, and the VPSF gateway and sensor devices were implemented using the open-source hardware Raspberry Pi 3 Model B+ running the Linux-based Raspbian OS.

Figure 3 depicts the VPSF implementation structure. The VPSF gateway was placed between the user device and multiple sensor devices. It communicates with the user device using the hypertext transfer protocol (HTTP) and communicates with the sensor devices using the CoAP. To implement the CoAP for the VPSF, we used the open-source software *libcoap*, a C-implementation of CoAP. We developed a monitoring application running on the VPSF gateway to examine the residual energy of each sensor device and the delay required to successfully store blocks. Each device was equipped with a wireless interface supporting IEEE 802.11 b/g/n and used a 100 Mbps data rate for data transmission.

For the experiment, we implemented four sensor devices, each initially set to have a different residual energy. Table 1 lists the initial settings for the residual energy of each sensor device. In the experiment, the size of user data varied from 100 MB to 1 GB. The block size was fixed as 1000 B, so the number of blocks per user data varied from 10^5 to 10^6. Each sensor device was equipped with a 32 GB SD card, and the storage space of each data volume was set equally to 20 GB. SPDYWND was set to 100 to consecutively transmit 100 blocks. To evaluate the VPSF performance, the VPSF experiment results were compared to legacy virtual private storage (legacy VPS), in which the gateway transmits the same number of blocks to each sensor device using the existing BlockS option of the CoAP.

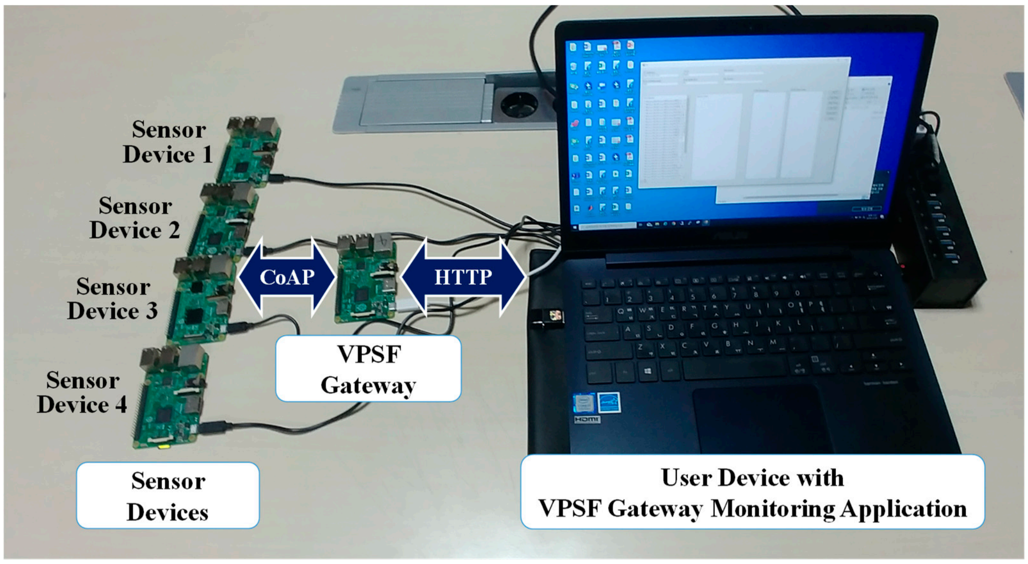


Figure 3. VPSF implementation structure.

Table 1. Initial setting for residual energy of each sensor device.

	Sensor Device 1	**Sensor Device 2**	**Sensor Device 3**	**Sensor Device 4**
Residual energy	8.14 Wh	16.28 Wh	24.42 Wh	32.56 Wh

Figure 4 depicts the changes in the lifetime of the sensor device with the lowest residual energy (Sensor Device 1) when the size of user data increases. The lifetime of the IoT local network is determined by the lifetime of the sensor device with the lowest residual energy. In this experiment, the user device was set to transmit user data continuously until the energy of Sensor Device 1 was depleted. Moreover, the sensor devices were set to overwrite user data to prevent the storage space of its data volume from filling up completely.

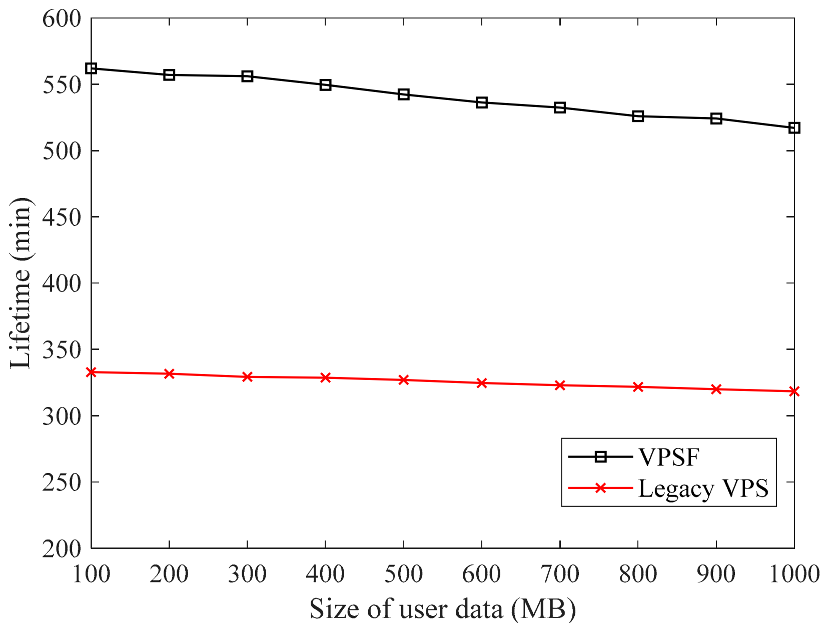


Figure 4. Lifetime of sensor device with the lowest residual energy.

The VPSF exhibited a longer lifetime than legacy VPS, owing to the VPSF's support of sensor device selection and the modified BlockS option. Specifically, Sensor Device 1 was used to store the blocks only after the residual energy of the other sensor devices reached that of Sensor Device 1. Furthermore, the lost blocks were retransmitted without any retransmission request messages in the VPSF. In contrast, in legacy VPS, the same number of blocks were stored in each sensor device whenever the user data were transmitted from the user device, and the retransmission was conducted with the retransmission request messages. In both cases, as the size of user data increased, the lifetime decreased; if the size of user data increased, the sensor device received and stored more blocks because the communication overhead caused by the transmission interval and the user data header decreased. On average, the VPSF exhibited a 65.85% longer lifetime than that of legacy VPS.

Figure 5 depicts the residual energy of each sensor device. In the experiment, the user device was configured to transmit 100 times to the VPSF gateway. Two sizes of user data (500 MB and 1 GB) were considered. In both cases, VPSF had a smaller difference in residual energy among sensor devices compared to legacy VPS because, unlike legacy VPS, the VPSF selected and used some of the sensor devices to store the blocks. Only Sensor Device 4 was used in the 500 MB case, and Sensor Devices 3 and 4 were used in the 1 GB case. The difference in residual energy among sensor devices can be mathematically represented by the fairness index F [35]. The fairness index is calculated as

$$F = \frac{\left(\sum_{i=1}^{n} e_i\right)^2}{n \sum_{i=1}^{n} e_i^2} \tag{2}$$

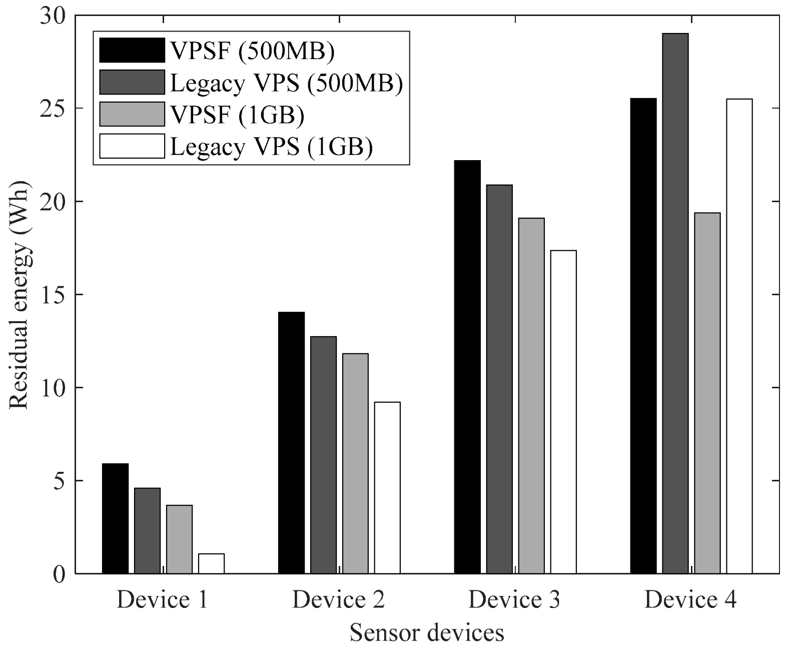


Figure 5. Residual energy of each sensor device.

The higher the fairness index, the smaller the difference in residual energy. Table 2 lists the fairness index of the VPSF and legacy VPS for each case. For 500 MB and 1 GB, respectively, the VPSF exhibited 7.58% and 19.77% higher fairness indexes than legacy VPS.

Table 2. Fairness index.

	VPSF (500 MB)	Legacy VPS (500 MB)	VPSF (1 GB)	Legacy VPS (1 GB)
Fairness index	0.83	0.77	0.82	0.68

Figure 6 depicts the variance in retransmission delay per user data when the size of user data changes. During the experiment, the block delivery ratio was 92.1%, on average, and 7.9% of blocks were lost and retransmitted. The VPSF employed the modified BlockS option, which enabled the server (VPSF gateway) to consecutively retransmit the lost blocks. The retransmission delay of the VPSF was 25.6% shorter than that of legacy VPS, on average. During the experiment, the retransmission delay increased as the size of user data increased—the larger the size of the user data, the higher the number of lost blocks.

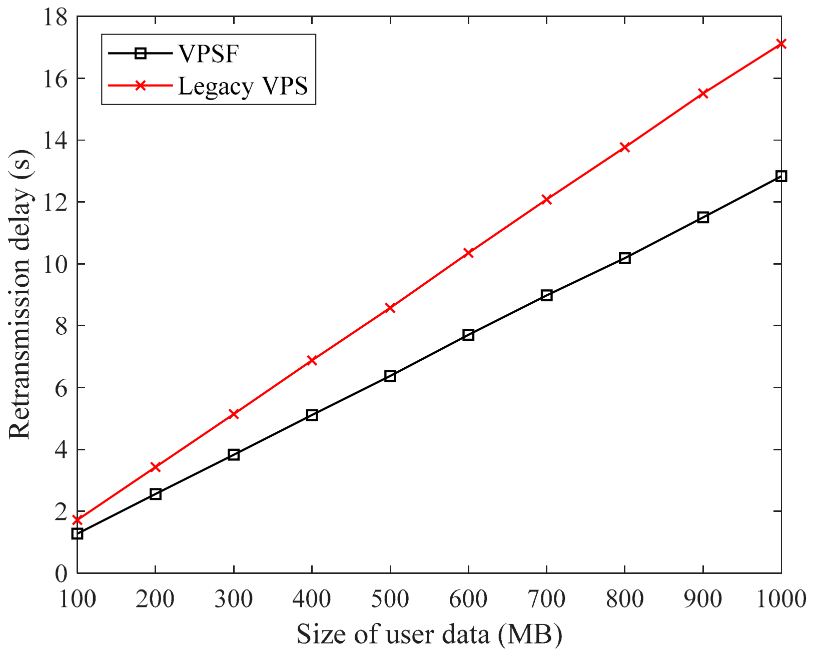


Figure 6. Retransmission delay based on size of user data.

4. Conclusions

In this paper, we presented a VPSF that uses the extra storage space of sensor devices in an IoT local network to store user data while guaranteeing the expected network lifetime. The VPSF gateway performs three operations to build and maintain virtual private storage: (1) virtual storage building, (2) sensor device selection, and (3) consecutive block retransmission. The first operation builds virtual private storage by integrating the extra storage space of sensor devices. The second operation selects a sensor device to store blocks based on the device's residual energy to guarantee the expected lifetime of the IoT local network. The third operation enables the VPSF gateway to retransmit the lost blocks without retransmission request messages to reduce communication overhead. To verify the feasibility of the VPSF, an experiment was performed using a specific implementation and the results were compared with legacy VPS. The results demonstrate that the VPSF exhibited a 65.85% longer lifetime and 25.6% shorter retransmission delay than those of legacy VPS.

Author Contributions: E.-J.K. conceived and designed the overall framework; H.-H.L. performed the open source-based implementation and experiment; J.-H.K. interpreted and analyzed the data; H.-H.L., J.-H.K., and E.-J.K. wrote the paper; E.-J.K. guided the research direction and supervised the entire research process. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2019R1I1A1A01059787).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Siddiqa, A.; Karim, A.; Gani, A. Big data storage technologies: A survey. *Front. Inf. Techol. Electron.* **2017**, *18*, 1040–1070. [CrossRef]
2. Dai, H.-N.; Wang, H.; Xu, G.; Wan, J.; Imran, M. Big data analytics for manufacturing internet of things: Opportunities, challenges and enabling technologies. *Enterp. Inf. Syst.* **2019**, 1–25. [CrossRef]
3. Kobusińska, A.; Leung, C.; Hsu, C.-H.; Raghavendra, S.; Chang, V. Emerging trends, issues and challenges in Internet of Things, Big Data and cloud computing. *Future Gener. Compt. Syst.* **2018**, *87*, 416–419. [CrossRef]
4. Yang, C.-T.; Chen, S.-T.; Cheng, W.-H.; Chan, Y.-W.; Kristiani, E. A Heterogeneous Cloud Storage Platform With Uniform Data Distribution by Software-Defined Storage Technologies. *IEEE Access* **2019**, *7*, 147672–147682. [CrossRef]
5. Google Drive. Available online: https://www.google.com/drive/ (accessed on 28 January 2020).
6. iCloud. Available online: https://www.icloud.com (accessed on 28 January 2020).
7. Dropbox. Available online: https://www.dropbox.com (accessed on 28 January 2020).
8. Park, J.K.; Kim, J. Big data storage configuration and performance evaluation utilizing NDAS storage systems. *AKCE Int. J. Graphs Comb.* **2018**, *15*, 197–201. [CrossRef]
9. Nachiappan, R.; Javadi, B.; Calheiros, R.N.; Matawie, K.M. Cloud storage reliability for big data applications: A state of the art survey. *J. Netw. Comput. Appl.* **2017**, *97*, 35–47. [CrossRef]
10. Singh, A.; Chatterjee, K. Cloud security issues and challenges: A survey. *J. Netw. Comput. Appl.* **2017**, *79*, 88–115. [CrossRef]
11. Mansouri, Y.; Toosi, A.N.; Buyya, R. Data storage management in cloud environments: Taxonomy, survey, and future directions. *ACM Comput. Surv.* **2017**, *50*, 1–51. [CrossRef]
12. Kishani, M.; Tahoori, M.; Asadi, H. Dependability analysis of data storage systems in presence of soft errors. *IEEE Trans. Reliab.* **2019**, *68*, 201–215. [CrossRef]
13. Xing, L.; Tannous, M.; Vokkarane, V.M.; Wang, H.; Guo, J. Reliability modeling of mesh storage area networks for Internet of Things. *IEEE Internet Things* **2017**, *4*, 2047–2057. [CrossRef]
14. Wu, Y.; Wang, F.; Hua, Y.; Feng, D.; Hu, Y.; Tong, W.; Liu, J.; He, D. I/O Stack Optimization for Efficient and Scalable Access in FCoE-Based SAN Storage. *IEEE Trans. Parallel Distrib.* **2017**, *28*, 2514–2526. [CrossRef]
15. Ammar, M.; Russello, G.; Crispo, B. Internet of Things: A survey on the security of IoT frameworks. *J. Inf. Secur. Appl.* **2018**, *38*, 8–27. [CrossRef]
16. Sheng, Z.; Tian, D.; Leung, V.C.M. Toward an energy and resource efficient internet of things: A design principle combining computation, communications, and protocols. *IEEE Commun. Mag.* **2018**, *56*, 89–95. [CrossRef]
17. Musaddiq, A.; Zikria, Y.B.; Hahm, O.; Yu, H.; Bashir, A.K.; Kim, S.W. A survey on resource management in IoT operating systems. *IEEE Access* **2018**, *6*, 8459–8482. [CrossRef]
18. Tang, J.; Sun, D.; Liu, S.; Gaudiot, J.-L. Enabling deep learning on IoT devices. *Computer* **2017**, *50*, 92–96. [CrossRef]
19. Ren, J.; Guo, H.; Xu, C.; Zhang, Y. Serving at the edge: A scalable IoT architecture based on transparent computing. *IEEE Netw.* **2017**, *31*, 96–105. [CrossRef]
20. Kurunathan, H.; Severino, R.; Koubaa, A.; Tovar, E. IEEE 802.15. 4e in a nutshell: Survey and performance evaluation. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1989–2010. [CrossRef]
21. Ceph. Available online: https://ceph.io/ (accessed on 28 January 2020).
22. Lustre. Available online: http://lustre.org/ (accessed on 28 January 2020).
23. Apache Hadoop. Available online: https://hadoop.apache.org/ (accessed on 28 January 2020).
24. Ghemawat, S.; Gobioff, H.; Leung, S.-T. The Google file system. In Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003; pp. 29–43.
25. Hu, P.; Dhelim, S.; Ning, H.; Qiu, T. Survey on fog computing: Architecture, key technologies, applications and open issues. *J. Netw. Comput. Appl.* **2017**, *98*, 27–42. [CrossRef]

26. 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*. [CrossRef]
27. Wang, T.; Zhou, J.; Chen, X.; Wang, G.; Liu, A.; Liu, Y. A three-layer privacy preserving cloud storage scheme based on computational intelligence in fog computing. *IEEE Trans. Emerg.* **2018**, *2*, 3–12. [CrossRef]
28. Moysiadis, V.; Sarigiannidis, P.; Moscholios, I. Towards distributed data management in fog computing. *Wirel. Commun. Mob. Comput.* **2018**, *2018*. [CrossRef]
29. Hao, Z.; Novak, E.; Yi, S.; Li, Q. Challenges and software architecture for fog computing. *IEEE Internet Comput.* **2017**, *21*, 44–53. [CrossRef]
30. Shelby, Z.; Hartke, K.; Bormann, C.; Frank, B. The Constrained Application Protocol (CoAP) (RFC 7252). Available online: https://tools.ietf.org/html/rfc7252 (accessed on 28 January 2020).
31. Bormann, C.; Shelby, Z. Block-Wise Transfers in the Constrained Application Protocol (CoAP) (RFC 7959). Available online: https://tools.ietf.org/html/rfc7959 (accessed on 28 January 2020).
32. Cao, Z.; Jin, K.; Fu, B.; Zhang, D. Speeding Up CoAP Block-wise Transfer. Available online: https://tools.ietf.org/id/draft-zcao-core-speedy-blocktran-00.html (accessed on 28 January 2020).
33. Libcoap. Available online: https://libcoap.net (accessed on 28 January 2020).
34. Libcoap Open Source. Available online: https://github.com/obgm/libcoap (accessed on 28 January 2020).
35. Kwon, J.-H.; Kim, E.-J. Asymmetric Directional Multicast for Capillary Machine-to-Machine Using mmWave Communications. *Sensors* **2016**, *16*, 515. [CrossRef] [PubMed]