A Space-Efficient Shredding-Based Method for Secure Data Deletion in Multi-Cloud Environments

Ali Khowilid Alqarni
Department of Computer Science
University of Kentucky
329 Rose Street
Lexington, KY 40506, USA
aal506@uky.edu

Zongming Fei
Department of Computer Science
University of Kentucky
329 Rose Street
Lexington, KY 40506, USA
fei@cs.uky.edu

***Abstract*—Cloud storage offers compelling scalability and convenience, but is also subject to significant risks, the greatest of which is exposure of sensitive information due to data replication, shared physical storage space, and lack of physical control. In particular, when a file is deleted in cloud computer systems, only index entries are removed, while data blocks containing information remain on the disks, leading to potential security risk for sensitive information. Encryption is the traditional protection mechanism, but it comes with significant detriments such as added processing overhead, complex key management, and a dependency on key secrecy. This paper presents a novel approach to secure data deletion by proposing a shredding-based approach, which breaks apart a file into two non-disclosing disjoint pieces, neither of which contains enough information to reveal the content of the original file. Each piece is stored in a different storage site in the multi-cloud environment. Unlike prior shredding approaches, this method allows to preserve the original file size to make it space efficient. Evaluations show that the processing overhead is comparable to the original shredding method, while the storage overhead is almost cut to half. The results from experiments demonstrate that it can be used in a practical system for secure file storage in multi-cloud environments.**

***Index Terms*—shredding, cloud computing, cloud storage, data security, file management**

I. Introduction

Cloud storage solutions have emerged as the mainstay of contemporary computing because of their versatility, scalability, and lower infrastructure costs. Individuals and businesses are increasingly dependent on cloud environments to monitor mission-critical data, access content from remote locations, and maintain service continuity. However, the advantages of cloud storage are accompanied by intrinsic threats, in particular, the vulnerability of sensitive information due to replication at several sites, shared physical storage space, and the lack of physical authority over the storage environment. The secure deletion of data in cloud storage presents a unique challenge because deletion of a file in computer systems only removes index entries and leaves intact disk blocks containing data. Traditional methods of data protection, in particular encryption, are widely applied to counter these threats. Although effective in theory, encryption also presents a number of operational challenges, such as performance overhead, dependence on successful key management, and a single point of failure if keys are compromised. These limitations have motivated researchers to explore other paradigms for data protection, ones that are not solely based on encryption.

Proactive shredding [1] addressed the data protection issue from a new perspective by proactively splitting the original data into pieces and then storing these pieces in different locations in the cloud. Each individual site does not contain complete information about the original data and thus compromising a single site will not expose the content. The approach developed an effective method for recovering the original data once all the pieces have been collected. It offers a new efficient way to manage data securely in the multi-cloud environment. One issue with the approach is that the size of each piece stored at any individual site is the same as that of the original data file. Thus, even if only two sites are used, the total storage required to store the original file will double the storage required to store the file as a single copy in the cloud.

In this paper, we propose a new space-efficient shredding method for secure data deletion in multi-cloud environments. Our goal is to keep the total storage requirement across all storage sites almost the same as storing the file as a single copy in the cloud. New shredding and weaving algorithms are designed using fixed masks and generating modifiers based on the values in the content itself, so that no external metadata need to be stored. The new method maintained the property of the original shredding approach that each individual site does not contain complete information about the original data and thus makes the data secure. Reducing the space requirement also leads to better performance in storing and retrieving files because less data need to be uploaded to or downloaded from each individual site. The proposed method has been evaluated both in a local environment and using the FABRIC distributed testbed, compared to our previous proactive shredding method [1] and the AES encryption [2] as a representative cryptographic baseline. Evaluations show that the processing overhead is comparable to the original shredding method, while the storage overhead is almost cut to half. The results from experiments in FABRIC demonstrate that it can be used in a practical system for secure file storage

in the multi-cloud environment.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the proposed space-efficient shredding method for the secure deletion of files in cloud storage. Section IV presents performance evaluation, and Section V concludes the paper.

II. Related Work

Many techniques have been developed to guarantee that data, once erased from the cloud, become truly irretrievable. These approaches aim to enhance the security of cloud computing while maintaining the trust of users, providers, and the underlying infrastructure.

One of the pioneering solutions was proposed by Mitra and Winslett, involving utilizing secure inverted index entries in order to impede data retrieval. The approach enforces rigorous deletion mechanisms and enhances the security of stored data [3]. Likewise, Luo proposed a masking overwriting technique that simulates data updates and thereby facilitates effective and secure deletion operations [4].

Cryptographic methods have also been used a lot to assist in the secure deletion process. Reardon et al. suggested utilizing dynamic B-Tree structures combined with encryption key management to avoid the recovery of deleted data by unauthorized means [5]. Common encryption models such as Advanced Encryption Standard (AES) and Attribute-Based Encryption (ABE) make data more secure but introduce system overhead and the issue of key distribution and management [2], [6].

In addition to encryption, partitioning methods such as network slicing have been employed to enhance data processing and resource management [7]. These methods have been helpful in distributed systems and 5G networks, especially when it comes to utilizing vertical slicing patterns to attain optimum performance and isolation [8].

Another aspect of concern in cloud storage is the storage space used in the cloud. The objective of CloudFC [9] is to save storage space in the cloud. It divides files into clusters based on access frequency and compress them with different compression ratios, with those accessed the least being compressed with a high compression ratio and those accessed the most not compressed at all. So, the overall storage requirement for the cloud is reduced. It shares a common goal with our work in trying to reduce the storage requirement in the cloud.

The most closely related is the proactive shredding approach for secure data deletion in the cloud [1]. It proposed a new shredding-based approach to managing data securely in the multi-cloud environment. One issue with the approach is that the total storage required increases linearly with the number of shreds or storage sites. This paper addresses the problem and extends the original shredding method by designing masks and modifiers in a new way so that the total storage required is almost the same as the size of the original file, instead of linearly increasing with the number of storage sites. This also brings the benefit of shorter upload and download times.

III. Methodology

A. Overview

The original shredding method [1] aims at enhancing security by splitting files into parts that are independently meaningless and stored in different sites, but can be easily combined to recover the original file. In particular, it developed two algorithms, shredding for splitting a file and weaving for recovering the file. For each unit of data (say a byte) A, shredding uses a randomly generated mask M and a randomly generated modifier R to split A into two pieces $D_0 = (A\&M) \oplus R$ and $D_1 = (A\& \sim M) \oplus R$. Weaving is much simpler by performing an exclusive OR operation $D_0 \oplus D_1$. The original file can be recovered even if the individual masks M and the modifiers R are not stored.

This work proposes a deterministic and space-efficient shredding method with the objective of saving cloud storage space. Instead of generating two data blocks that are of equal size compared with the original data unit A, our method uses a pair of data units at a time, and generates two data blocks with each block being the same size as that of the original data unit. Therefore, the total storage required is the same as the original file.

B. Shredding Process

We use byte as the data unit to present the shredding process, even though a larger data unit can be used. The shredding process operates on a pair of two bytes S_0 and S_1 at a time, until the end of the input file. Each byte is broken down into individual bits denoted as $S_{0,i}$ and $S_{1,i}$, where $i \in [0, 7]$.

We use two fixed masks M_E and M_O to extract the values of even bits and odd bits, respectively, with $M_E = \texttt{10101010}$, $M_O = \texttt{01010101}$. Notice $M_O = \sim M_E$.

Each time, the shredding process takes in two bytes S_0 and S_1 from the input file and transforms them into two bytes D_0 and D_1. All D_0's become one shred and all D_1's become the other shred. These two shreds will be stored at different cloud sites. The shredding algorithm combines the even bits of S_0 with the odd bits of S_1, and performs an XOR operation ($\oplus$) on it with the bit pattern determined by function f. D_1 is generated similarly by combining the odd bits of S_0 and the even bits of S_1. The shredding process can be represented by

$$D_0 = ((S_0 \ \& \ M_E) \ | \ (S_1 \ \& \ M_O) \ ) \oplus \ f(S_{1,0}, S_{0,1})$$

and

$$D_1 = ((S_0 \ \& \ M_O) \ | \ (S_1 \ \& \ M_E) \ ) \oplus \ f(S_{0,0}, S_{1,1})$$

The modifier function $f(x, y)$ produces an 8-bit output from the two input bits as shown in Table I. The first two bits of the output need to be 00 to ensure that the process is reversible to recover the original data from D_0 and D_1, while the other six bits can be designed without any special restrictions.

Fig. 1 illustrates the detailed calculations for generating D_0 using the original data S_0 and S_1. D_1 can be generated in a similar way.

TABLE I
MODIFIER GENERATION FUNCTION $f(x, y)$

(x, y)	$f(x, y)$
$(0, 0)$	00100101
$(0, 1)$	00011110
$(1, 0)$	00110010
$(1, 1)$	00111011

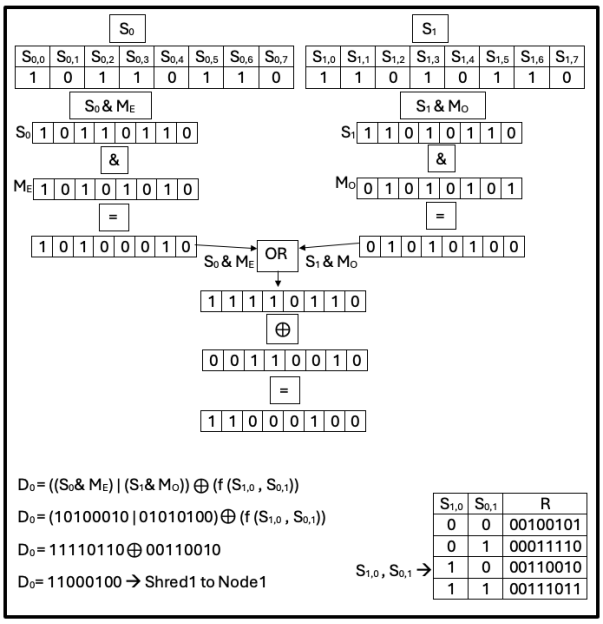

Fig. 1. Shredding Process: Generating D_0

C. Weaving Process

The weaving process reconstructs the original data S_0 and S_1 from the two shredded outputs D_0 and D_1. Again we will use the same modifier function $f(x, y)$ defined in Table I. We can use the following weaving method to recover S_0 and S_1.

$$S'_0 = ((D_0 \oplus f(D_{1,0}, D_{1,1}))\&M_E) \mid ((D_1 \oplus f(D_{0,0}, D_{0,1}))\&M_O)$$

$$S'_1 = ((D_0 \oplus f(D_{1,0}, D_{1,1}))\&M_O) \mid ((D_1 \oplus f(D_{0,0}, D_{0,1}))\&M_E)$$

It can be proved that $S'_0 = S_0$ and $S'_1 = S_1$. Notice $D_{0,0} = S_{0,0}, D_{0,1} = S_{1,1}, D_{1,0} = S_{1,0}$ and $D_{1,1} = S_{0,1}$. Fig. 2 shows the reconstruction of S_0 using the shreds D_0 and D_1. S_1 can be reconstructed in a similar way.

D. Padding

Since the shredding process takes a pair of bytes every time, we need to deal with the case in which the total number of bytes is odd. We can solve the issue by adding padding bytes at the end. If the number of bytes is odd, we add an extra zero-filled padding byte 00000000. In order to distinguish whether a padding byte is added or not, two additional bytes are added. If a zero filled padding byte is added, another two bytes of 10000000 00000000 will be added. If a zero filled padding

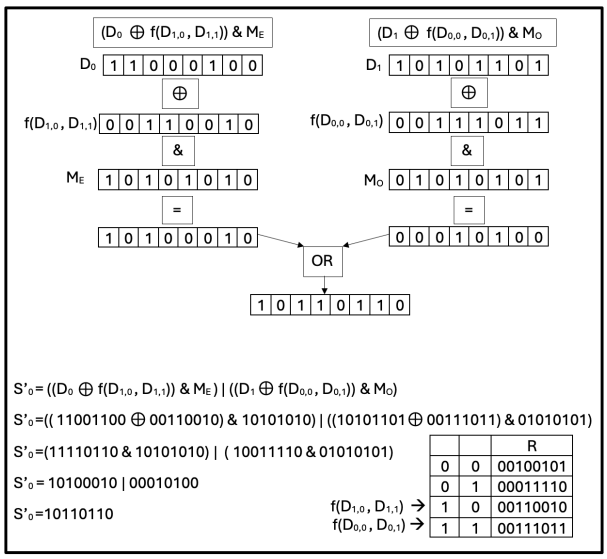

Fig. 2. Weaving Process: Reconstructing S_0

byte is not added, two more bytes of 00000000 00000000 will be added. In the worst case, three extra bytes will be added before shredding. In the weaving process, two bytes will be removed if the last two bytes in reconstructed data are 00000000 00000000; otherwise three bytes will be removed.

E. An example

We finish this section by providing an example in Fig. 3, which demonstrates how a text file is shredded into two independent fragments, Shred1 and Shred2, each stored on a separate cloud node. The weaving process retrieves both shreds and successfully reconstructs the original file, verifying the correctness of the method.

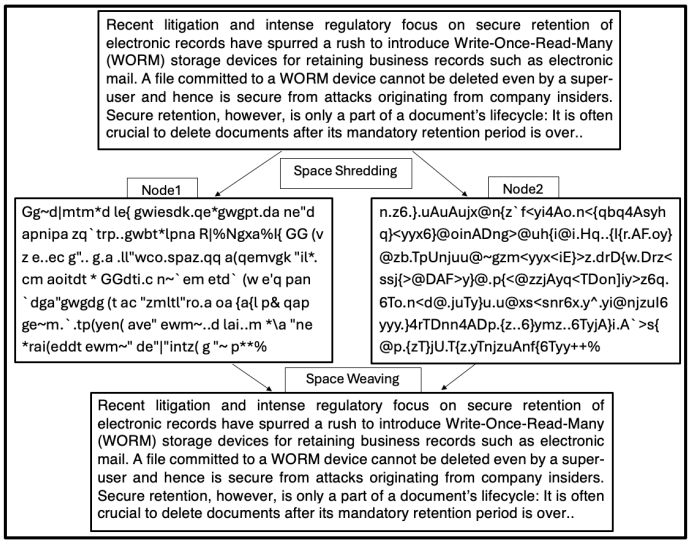

Fig. 3. An example of shredding and weaving a text file across two nodes

IV. Performance Evaluation

This section evaluates the performance of the proposed space-efficient shredding and weaving method, comparing it with the original shredding and weaving method, and traditional AES encryption/decryption algorithms as a cryptographic baseline. Initial simulations were conducted in a local environment using Python. To further validate the method under realistic conditions, we extended the evaluation to a distributed environment using the FABRIC testbed, which allows us to distribute file shreds across geographically separated nodes, simulating practical multi-cloud storage scenarios.

A. Performance of Shredding and Weaving Algorithms

We first focused on evaluating the performance of the shredding and weaving algorithms, including both the original shredding method (called shredding/weaving) and the proposed space-efficient approach (called space shredding/space weaving). The standard AES encryption and decryption algorithms were also included as a baseline for comparison. File sizes ranging from 1 MB to 1 GB were tested.

1) Shredding vs. Space Shredding vs. Encryption

Fig. 4 compares the performance of shredding, space shredding, and encryption for files with sizes ranging from 1 MB to 1 GB, doubling each time. The x-axis is on logarithmic scale, and the y-axis is on linear scale, showing the time in seconds. As the file size increases, the time increases for all three methods. However, encryption takes more time than both shredding methods. For example, for a 1 GB file, encryption takes 5.357 seconds, while normal shredding takes 3.619 seconds, and space shredding takes 4.002 seconds. Normal shredding is faster because it uses simple operations like AND and XOR. Space shredding is slightly slower due to the extra steps of mapping bits.

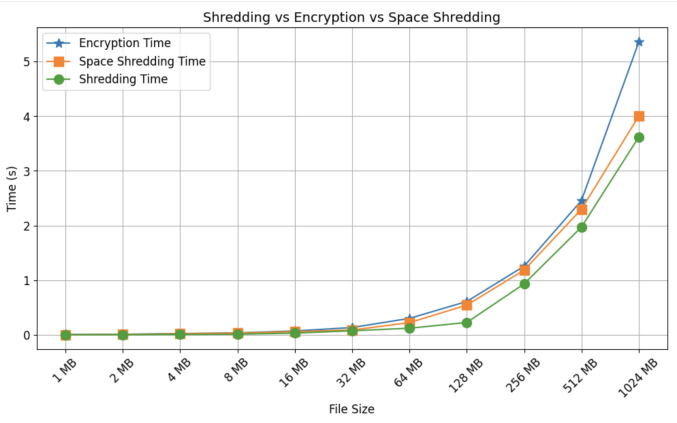

Fig. 4. Shredding vs. Space Shredding vs. Encryption processing times for files of different sizes in local environment (Time in Seconds)

2) Weaving vs. Space Weaving vs. Decryption

Fig. 5 shows the time needed to reconstruct files using normal weaving, space weaving, and decryption, for file sizes from 1 MB to 1 GB. As file size increases, decryption time grows much faster than weaving. For example, for a 1 GB file, decryption takes 5.293 seconds, while normal weaving takes 0.990 seconds, and space weaving takes about 3.083 seconds. Normal weaving is the fastest since it uses basic XOR operations. Space weaving is slower than normal weaving due to extra processing, while decryption is the slowest due to more complicated operations.

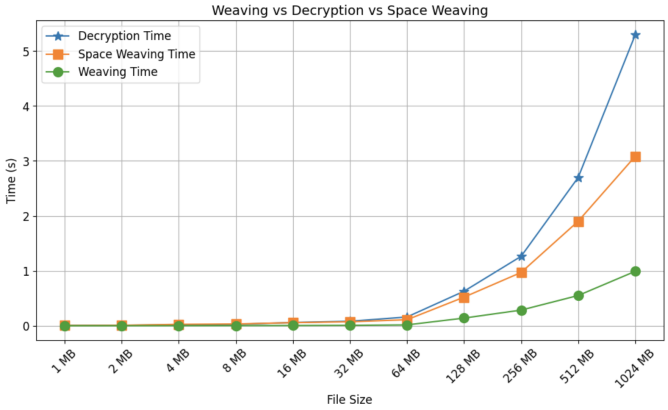

Fig. 5. Weaving vs. Space Weaving vs. Decryption processing times for files of different sizes in local environment (Time in Seconds)

B. Performance Evaluation in Storage and Retrieval Using FABRIC

After testing the effectiveness of space shredding algorithms in a local environment, we expanded our study using a cloud-based setting with the FABRIC testbed [10]. FABRIC offers a distributed testbed that simulates real-world cloud operations, such as storing, retrieval, and deletion of files across distributed nodes. To avoid inconsistency and to facilitate analysis, all measurements performed within this assessment were limited to one node (Node1), although the shredding logic inherently includes multiple nodes. We compare the normal shredding method and the space shredding method. Two primary scenarios were tested: storing a file (shredding + upload), and retrieving a file (download + weaving). File sizes range from 1 MB to 256 MB, and operations were executed through the JupyterHub interface on FABRIC, which offers a standard 1 GB storage allocation.

1) Storing a File

This scenario includes both shredding and uploading procedures. As illustrated in Fig. 6, space shredding consistently outperforms traditional shredding in terms of upload time, especially as file size increases. For example, storing a 256 MB file takes approximately 18.1 seconds with normal shredding, while space shredding completes the same task in just 10.1 seconds — a performance gain of over 44.3%.

2) Retrieving a File

In this scenario, we evaluate the total time required to retrieve and reconstruct a file from the cloud. As illustrated

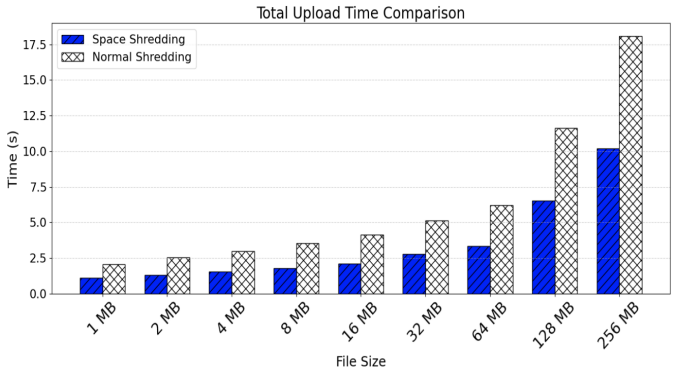

Fig. 6. Performance comparison for storing a file (Shredding + Upload)

in Fig. 7, the download and reconstruction times increase with file size for both methods. However, space weaving consistently outperforms normal weaving across all file sizes. The performance advantage becomes more significant for larger files. For instance, retrieving a 256 MB file takes about 11.18 seconds using normal weaving, while space weaving requires only about 6.86 seconds, which is around 39% faster than normal weaving.

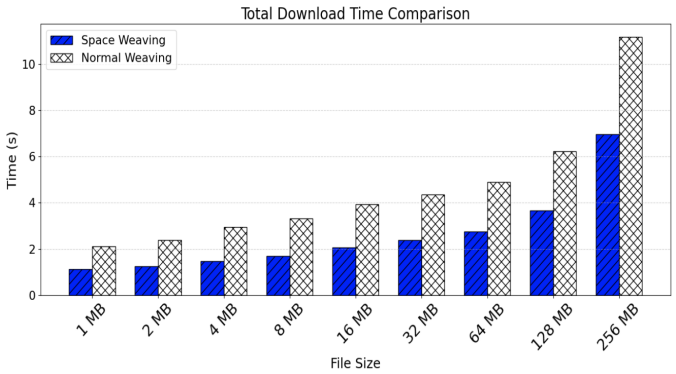

Fig. 7. Performance comparison for file retrieval (Weaving + Download)

3) Deleting a File

The deletion process in both approaches was found to be quick and consistent, as it mainly involves issuing deletion commands against one or more nodes. The time taken for deletion is negligible and the difference is nearly imperceptible between the two approaches.

V. Concluding Remarks

This paper proposed a space-efficient shredding scheme to reduce the storage requirement for secure data deletion in multi-cloud environments. Evaluations in the local environment demonstrated that the proposed method is efficient, compared to encryption/decryption-based methods. Experimental evaluations using FABRIC showed that the space-efficient method also leads to performance improvement in storing and retrieving operations because less data need to be uploaded to and downloaded from the cloud. As the overhead introduced by the space shredding method is low, it can be used in a practical cloud system for secure data management.

References

[1] A. Alqarni, E. Fei, and Z. Fei, "A Proactive Shredding Approach for Secure Deletion of Content in the Multi-Cloud Environment," in Proceedings of the 2025 34th IEEE International Conference on Computer Communications and Networks (ICCCN), Tokyo, Japan, 2025, pp. 1–6, doi: https://doi.org/10.1109/ICCCN65249.2025.11133969.

[2] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES) (FIPS PUB 197)," U.S. Department of Commerce, Nov. 2001. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.197.

[3] S. Mitra and M. Winslett, "Secure deletion from inverted indexes on compliance storage," ACM, 2006. [Online]. Available: https://doi.org/10.1145/1179559.1179572

[4] Y. Luo, M. Xu, S. Fu, and D. Wang, "Enabling assured deletion in the cloud storage by overwriting," ACM, 2016. [Online]. Available: https://doi.org/10.1145/2898445.2898447

[5] J. Reardon, H. Ritzdorf, D. Basin, and S. Capkun, "Secure data deletion from persistent media," ACM, 2013. [Online]. Available: https://doi.org/10.1145/2508859.2516699

[6] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP'07)*, Berkeley, CA, USA, 2007, pp. 321–334, doi: https://doi.org/10.1109/SP.2007.11.

[7] Q. Li, G. Wu, A. Papathanassiou, and U. Mukherjee, "An end-to-end network slicing framework for 5G wireless communication systems," IEEE, 2016. [Online]. Available: https://arxiv.org/pdf/1608.00572

[8] F. Moscatelli, G. Landi, N. Ciulli, A. Tzanakaki, and S. Prior, "Demo: Service-driven management of Vertical Services in heterogeneous network slices," in *Proceedings of the 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, Athens, Greece, 2021, pp. 1–3, doi: https://doi.org/10.1109/MeditCom49071.2021.9647667.

[9] H. Yahyaoui and S. Moalla, "CloudFC: Files Clustering for Storage Space Optimization in Clouds," in *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg, Luxembourg, 2016, pp. 193–197, doi: https://doi.org/10.1109/CloudCom.2016.0042.

[10] FABRIC, https://portal.fabric-testbed.net/about/about-fabric.

[11] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. "FABRIC: A national-scale programmable experimental network infrastructure." IEEE Internet Computing 23, no. 6 (2019): 38–47, doi: https://doi.org/10.1109/MIC.2019.2958545.