ADAPTIVE DATA MANAGEMENT FOR CLOUD-BASED WIRELESS MESH NETWORKS

YANG SHENGTAO

M.Phil

The Hong Kong Polytechnic University

2017

The Hong Kong Polytechnic University

Department of Computing

Adaptive Data Management for Cloud-based Wireless Mesh Networks

YANG Shengtao

A thesis
submitted in partial fulfilment of the requirements
for the degree of

Master of Philosophy

April 2016

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

YANG Shengtao (Name of student)

Abstract

In recent years, there has been considerable interest in developing and using wireless mesh networks. Compared to wireless local area networks, wireless mesh networks are more flexible because wireless mesh routers are interconnected by wireless links. In addition, they can be much easier to install and maintain, especially in environments where cables are difficult to install. Inspired by cloud computing, the aim of this project is to investigate a cloud-based wireless mesh network with adaptive data storage functions for storing data dynamically and flexibly in a wireless environment. In particular, a person-based adaptive data management scheme and a group-based adaptive data management scheme were designed. The person-based adaptive data management scheme seeks to provide upload/download functions for mesh clients, and adaptively moves files along with the movements of the owner to enhance access efficiency. The group-based adaptive data management scheme seeks to determine how the data resources/files should be stored and replicated in the wireless mesh routers such that the overall access cost can be minimized. Both a heuristic algorithm and a genetic algorithm were investigated. To support a cloud-based wireless mesh network, a distributed file system called MeshFS was also developed. A key technical challenge is to develop a lightweight software system that can be implemented over memory-limited wireless mesh network environments. MeshFS integrates scattered storage resources from wireless mesh routers to provide a mountable file system with fault-tolerant capabilities and cloud computing-like storage functions.

Publications

Shengtao Yang, Henry C. B. Chan, Patrick P. Lam, and Peter H. J. Chong. "A Cloud-based Wireless Mesh Network with Adaptive Data Storage Functions". In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2. 2015, pp. 540-545. [1]

Shengtao Yang, Henry C. B. Chan, Patrick P. Lam, and Peter H. J. Chong, "MeshFS: A Distributed File System for Cloud-based Wireless Mesh Network". submitted to *Journal of Systems and Software*.

Shengtao Yang, Henry C. B. Chan, Patrick P. Lam, and Peter H. J. Chong, "A Group-based Adaptive Data Management Mechanism for Cloud-based Wireless Mesh Networks". pending submission.

[1] Best Student Paper Award of The 2015 IAENG International Conference on Communication Systems and Applications

Acknowledgements

Firstly, I would like to express my sincerest gratitude to my chief supervisor, Dr. Henry C. B. Chan, for offering me the opportunity to conduct this research study, as well as for his teaching, inspiring guidance, and generous advice throughout.

I would also like to express my sincere gratitude to P2 Mobile Technologies Limited and the Innovation and Technology Fund of the Hong Kong Special Administrative Region Government, for their sponsorship and generosity in providing the opportunity for this research study.

I would like to express my appreciations to all of the professors from the Department of Computing, who have taught and guided me throughout my undergraduate and postgraduate studies.

I would like to thank Dr. Patrick Lam, Justin Yu, Mond Wan, and other staff members at P2MT, all of those who have kindly provided me with guidance and assistance with wireless mesh network equipment and technologies.

I would also like to thank my lab mates: Li Yingxiong, Dou Yi, Lau Shiu Fung, Ho Yik Him, and others, for their support, sharing, encouragement and company.

I would like to give special thanks to my family for providing me with the opportunity for a great education as well as for their ongoing support, which has allowed me to make the progress that I have to this day.

I would like to dedicate this thesis to all of the people who have guided, instructed, and helped me with this research study, which could not have been accomplished without all of your support. Thank you.

Contents

Abstract vii

Publications ix

Acknowledgements xi

List of Figures xvii

List of Tables xix

1 Introduction 1
- 1.1 Research Objectives . . . 2
- 1.2 Thesis Structure . . . 3

2 Literature Review 5
- 2.1 Wireless Mesh Network . . . 5
- 2.2 Cloud Computing . . . 8
- 2.3 Distributed File System . . . 12

3 Person-based Adaptive Data Management Mechanism 15
- 3.1 System Design & Architecture . . . 15
- 3.2 Hardware & Prototype . . . 17
- 3.3 File System . . . 19
  - 3.3.1 Storage Function with Linux File System . . . 19
  - 3.3.2 Storage Function with A Distributed File System . . . 22
- 3.4 Implementation . . . 23
  - 3.4.1 Web UI for File Uploading/Management . . . 23
  - 3.4.2 Agent Program for Monitoring the Mobility of Clients and Making Decisions on the Movement of Files . . . 25
- 3.5 Conclusion . . . 29

4 MeshFS: A Distributed File System for Cloud-based Wireless Mesh Network 31
- 4.1 Objectives & Features . . . 31

4.2 System Design & Architecture 33
4.3 Implementation 35
4.3.1 Messaging 35
4.3.2 Client 36
4.3.3 Server 39
4.3.4 Metaserver 40
4.3.5 Syscalls 45
4.4 Demonstration 51
4.5 Evaluation & Limitation 55
4.6 Conclusion 60

5 Group-based Adaptive Data Management Mechanism 63
5.1 E-book Scenario 63
5.2 Model & Algorithm 67
5.2.1 Model 67
5.2.2 Problem Definition 68
5.2.3 Proposed Solution & Algorithm 68
5.3 Simulation & Evaluation 71
5.3.1 Influence of a Single Parameter 72
5.3.2 Influence of a Combination of Parameters 75
5.3.3 Comparison to the Brute-Force Solution 78
5.3.4 Comparison to Genetic Algorithm 82
5.4 Conclusion 91

6 Conclusion & Future Work 93

References 95

List of Figures

2.1 Wireless Mesh Network 5

3.1 Architecture of a Cloud-based Wireless Mesh Network 16
3.2 Person-based Adaptive Data Management Mechanism 17
3.3 Router model MeshRanger from P2MT 17
3.4 Wireless Mesh Network Formed by MeshRangers 18
3.5 One of the Possible Topologies of the WMN Backbone 20
3.6 UML Diagram for Data Location Index 21
3.7 Schema of Table Client in MySQL 21
3.8 Schema of Table File in MySQL 22
3.9 Schema of Table Router in MySQL 22
3.10 Schema of Table FileStorage in MySQL 22
3.11 Web UI for Person-based Adaptive Data Management System . 24
3.12 Work Flow of the File Transfer Operation 28

4.1 Sample Architecture of MeshFS 34
4.2 Working Sequence of a MeshFS Server 39
4.3 ER Diagram of a MeshFS Metaserver Database 41
4.4 Sequence Diagram of the Backup Function 44
4.5 General Working Sequence of Messaging 45
4.6 Sequence Diagram of syscall *read()* 47
4.7 Sequence Diagram of syscall *write()* 48
4.8 Sequence Diagram of syscall *rename()* 49
4.9 Sequence Diagram of syscall *unlink()* 50
4.10 Demo Website with Multimedia Resources Hosting on MeshFS . 51
4.11 MeshFS Reading Data when Video is Playing 52
4.12 Data Lost without Enabling Backup Function 52
4.13 Video Stopped at the Corrupting Point without Backup 53
4.14 Video Continued at the Corrupting Point with Backup 53
4.15 Corrupted Data Recovered by MeshFS with Backup/Recovery Enabled 54
4.16 Summary of MeshFS Demo 54
4.17 Relationship between Data Size and Time in MeshFS 55

4.18 Relationship between Data Size and Transfer Speed in MeshFS . 56
4.19 Comparison of the Flow of Data when Playing Videos in MeshFS with the Backup Enabled/Disabled 57
4.20 Comparison of the Flow of Data when Playing Videos of Different Sizes in MeshFS with the Backup Enabled 57
4.21 Relationship between Recovery Time and Size of Video Files . . 58
4.22 Relationship between Recovery Speed and Size of Video Files . . 59

5.1 E-book Scenario 63
5.2 E-book Scenario: Cost for 1 Student 64
5.3 E-book Scenario: Cost for 100 Students 65
5.4 E-book Scenario: Saving $99c_1$ Cost after the File is Moved . . . 66
5.5 E-book Scenario: Multiple Copies of the File are Spread in the WMN 66
5.6 Relationship between Cost Reduction and Number of Routers . 73
5.7 Relationship between Cost Reduction and Link Bandwidth . . . 73
5.8 Relationship between Cost Reduction and Number of Files . . . 74
5.9 Relationship between Cost Reduction and File Size 75
5.10 Relationship between Cost Reduction and Number of Requests . 76
5.11 Relationship between Cost Reduction and Number of Routers with Different Numbers of Requests 76
5.12 Relationship between Cost Reduction and Number of Requests with Different Numbers of Routers 77
5.13 Surface Diagram of Cost Reduction, Number of Requests, and Number of Routers 78
5.14 Relationship between Cost and Number of Routers in Comparison with the Brute-Force Algorithm 79
5.15 Relationship between the Effectiveness Ratio and the Number of Routers in Comparison with the Brute-Force Algorithm 80
5.16 Relationship between Computation Time and Number of Routers in Comparison with the Brute-Force Algorithm 81
5.17 Relationship between the Efficiency Ratio and the Number of Routers in Comparison with the Brute-Force Algorithm 81
5.18 Relationship between Cost and Number of Routers in Comparison with the Genetic Algorithm 86
5.19 Relationship between the Effectiveness Ratio and the Number of Routers in Comparison with the Genetic Algorithm 87
5.20 Relationship between Cost and Number of Requests in Comparison with the Genetic Algorithm 88

5.21 Relationship between the Effectiveness Ratio and the Number of Requests in Comparison with the Genetic Algorithm 88
5.22 Relationship between Computation Time and Number of Routers in Comparison with the Genetic Algorithm 89
5.23 Relationship between the Efficiency Ratio and the Number of Routers in Comparison with the Genetic Algorithm 89
5.24 Relationship between Computation Time and Number of Requests in Comparison with the Genetic Algorithm 90
5.25 Relationship between the Efficiency Ratio and the Number of Requests in Comparison with the Genetic Algorithm 91

List of Tables

2.1 Major Cloud Styles in Industry and Adopters 11
2.2 Summary & Comparison of Popular DFSs 12

4.1 List of Request Messages 37
4.2 List of Syscalls 38
4.3 List of Operation Calls 40
4.4 Comparison between the Normal Mode and the "big_writes" Mode for *write()* 60

1 Introduction

During the past decade, wireless networks went through rapid growth of development. In comparison with traditional wired networks, wireless networks have better flexibility, configurability, and stability. In wireless networks, users can move around freely within the network range, and the network can support more concurrent users without being limited by the number of cable ports. Wireless networks have significantly promoted the usage of mobile devices, services, and applications. Currently, the most popular types of wireless networks are Wireless Local Area Networks (WLAN) and Cellular Networks. The most popular type of WLAN that follows IEEE 802.11 standards series is Wi-Fi. Wireless Mesh Network is another popular type of wireless network, which refers to a wireless network formed by radio nodes in mesh topology. Compared to traditional WLANs, wireless mesh networks have even better flexibility, since WLAN access points are normally wired, costing extra fees and time for the cabling work required. Wireless mesh network access points, which are also known as mesh routers, are wirelessly connected to one another. Hence they come at very low installation and maintenance cost, and provide better coverage in environments where it is difficult to place wires. Wireless mesh networks also come with other advantages, such as being self-configuring, self-healing, having easy extensibility, etc.

Another technology that has recently attracted a great deal of attention is Cloud Computing. In cloud computing, cloud resources are shared by users in order to access resources and services remotely, and the resources can be utilized on demand to improve effectiveness. Cloud vendors have seen very high growth rates in recent years, and offer different models of service such as Infrastructure as a service (IaaS), platform as a service (PaaS), Software as a Service (SaaS), etc. Cloud computing has become an interesting and valuable topic in both academic and industrial research.

The purpose of this research project is to integrate wireless mesh network and cloud computing to build a cloud-based wireless mesh network and to develop an adaptive data management system. This project is co-supervised by the Department of Computing, The Hong Kong Polytechnic University (PolyU), and P2 Mobile Technologies (P2MT) under the Teaching Company Scheme (TCS).

1.1 Research Objectives

To introduce new features to existing wireless mesh networks to enhance their functionalities, this research aims to investigate and design a cloud-based wireless mesh network with adaptive data management mechanisms. In general, the main objectives of this research are:

- To design the system architecture for a cloud-based wireless mesh network
- To design a person-based adaptive data management mechanism for the cloud-based wireless mesh network
- To design a group-based adaptive data management mechanism for the cloud-based wireless mesh network
- To evaluate the person-based and group-based adaptive data management mechanisms

The purpose of this research study is to design and develop a systematic method of adaptive data management for a cloud-based Wireless Mesh Network (WMN) to achieve efficient data access. The study mainly focuses on two functionalities: 1. Person-based data management: tracking a user's location in the WMN and maintaining the user's personal data in the nearest mesh router to follow up on the user's movement among different mesh routers. 2. Group-based data management: dynamically allocating data to the mesh router(s) where the data will be used most frequently to improve utilization rate and reduce cost.

To support previous functionalities, there are some assumptions:

1. The WMN not only provides network access and communication services, but also supports file uploading and storing on the network cloud.
2. The mesh routers should contain second memory devices (e.g. hard disks, etc.) to support file storage and file exchange functions.

1.2 Thesis Structure

Chapter 2

This chapter aims to conduct a comprehensive background and literature review of related topics for the research. The relative topics include: wireless mesh network, cloud computing, and distributed file system.

Chapter 3

This chapter presents the first part of this research: the person-based adaptive data management mechanism. First, this chapter discusses the system design and architecture of the newly introduced cloud-based wireless mesh network. Then it introduces the hardware platform used, as well as the prototype implemented in this research. Later, the chapter discusses file system and implementation details of the prototype.

Chapter 4

In Chapter 3, it has been discussed that both person and group-based data management mechanisms can be used on a distributed file system (DFS) to achieve better flexibility and configurability. However, currently existing DFS solutions mainly focus on powerful servers, which the mesh routers cannot compete with in terms of computing capacities. This chapter presents MeshFS, a distributed file system specifically designed for wireless mesh networks as the second part. All aspects including features, design & architecture, implementation, demonstration, evaluation & limitations will be discussed in detail in this chapter.

Chapter 5

This chapter discusses the third part of this research: the group-based adaptive data management mechanism. The chapter first describes a situation called an e-book scenario, which will be suitable to apply this mechanism to. Then a

mathematical model and algorithm will be presented, followed by simulations to evaluate its performance.

Chapter 6

This final chapter concludes all of the work of this research project and outlines future work.

Literature Review

2

2.1 Wireless Mesh Network

A Wireless Mesh Network (WMN) is a wireless communication network formed by multiple mesh routers and mesh clients [1, 2, 4] (Fig. 2.1). Mesh routers, also known as Mesh Access Points (MAPs), are interconnected by mesh configurations and responsible to perform data exchanging and routing functions. Mesh clients (e.g. mobile phones, notebooks) are connected to a MAP to exchange data and access network services. Similar to traditional wireless local area network (WLAN), WMN provides data communication and network services to wireless clients, however mesh routers in WMNs are connected wirelessly instead of being linked by wired cables. Hence, compared to traditional WLAN network infrastructure, a WMN has better flexibility and fewer effects in terms of limitations/constraints from the environment.

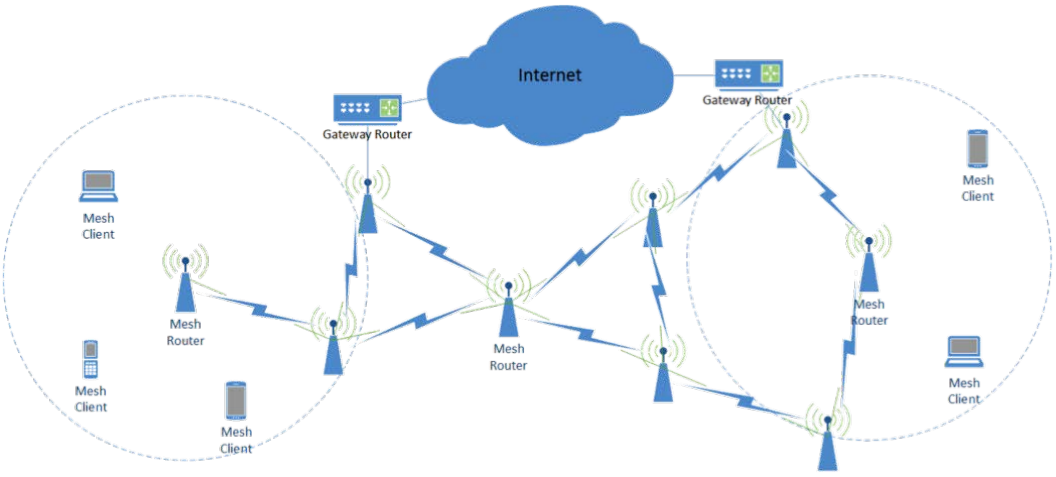

Fig. 2.1: Wireless Mesh Network

Mesh routers can be interconnected as a self-organized and self-configured ad hoc network in a peer-to-peer way, or establish a WMN backbone by following current wireless network standards, such as the IEEE 802.11 standards series. Currently, IEEE 802.11-based WMN is the most popular type of WMN [39], which operates on 2.4, 3.6, 5, and 60 GHz frequency bands. Following the launch of the first IEEE 802.11 standard in 1997, there is now a set of standards for different requirements and conditions, namely 802.11a, 802.11b, 802.11g and 802.11n. The latest standard, IEEE 802.11n operates over both 2.4 GHz

and 5 GHz bands and supports a theoretical data rate of up to 600 Mbit/s using Multi-Input & Multi-Output (MIMO) antenna technology [37]. The next generation of IEEE 802.11 standard is IEEE 802.11ac, expected to be released in 2014. It can support an extremely high data rate of up to 6.93 Gbit/s using 8 MIMO channels [44].

In addition to providing internal network services among mesh routers and mesh clients, WMNs can also be easily integrated with other types of networks through gateway/bridge functionalities. For a WMN that has at least one mesh router connected to an external network (e.g. Internet), this/these mesh router(s) will perform as a network gateway so that other mesh routers and clients can access external network resources. Devices can connect to a WMN either by using a wireless network interface controller, or a traditional Ethernet interface. This feature gives WMNs significant scalability and extensibility.

For research work about the basics of WMN, [17] and [33] which addressed routing issues in a WMN and [8] which studied a scheduling problem for WMN. [30, 36, 42] studied more fundamental and architectural issues related to WMN. The aforementioned work will provide useful references and valuable insight into the design of the proposed cloud-based WMN. Furthermore, it will take user mobility over a WMN into consideration (e.g., see [7]).

While research on cloud-based wireless mesh networks together with adaptive data management is a relatively new topic, there are related works on data caching for WMN. The following provides some examples. In the early stages, Imielinski and Badrinath addressed and predicted the challenges of data management in wireless computing [27, 28]. Jin and Wang studied and explored the optimality of content and service replication strategies in multi-hop WMNs [29]. Nandan, Das, Pau, Gerla, and Sanadidi proposed SPAWN, a content delivery and sharing cooperative strategy for vehicular ad hoc wireless networks [35]. Chow, Leong, and Chan discussed and presented a group-based cooperative caching management in mobile environments [9, 10, 11].

Das, Pucha and Hu designed and evaluated a MeshCache system, which is a hop-by-hop cooperative caching system for WMN [12]. The hop-by-hop caching and fetching mechanism takes user locations into consideration in order to enhance throughput (e.g., to fetch content from the nearest cache). This paper proposed hop-by-hop cooperative caching to improve network performance by exploiting locality existing in the traffic of a wireless mesh network. The hop-by-hop caching mechanism is adopted to replace the traditional end-to-end

caching mechanism. Assume there is a route path S-A-B-D, the hop-by-hop caching scheme will break up this single path into S-A, A-B, B-D, and cache the data at A and B as well as node D to increase content availability.

Denko and Nkwe proposed an efficient caching scheme for WMN called CacheRescue [14]. In CacheRescue, valid but evicted data items can be stored in mesh routers using extendable storage space. Based on a cluster architecture, Mesh Routers (MRs) function as Cluster Heads (CHs) for Mesh Clients (MCs). A CacheRescue Database (CRDB) is set up in CH so that data from MC?s local cache can be cached in CRDB. The storage size can be adjusted dynamically based on number of MCs as well as their cache sizes. The following equation shows how the MR?s current CacheRescue size can be estimated based on the expected cache size of other nodes:

$$E[C|C_i] = \frac{C_{total} - C_i}{N - 1}$$

where C_{total} is the total number of caches' size, N is the number of CHs, and C_i is the cache size.

Wu and Huang proposed a cooperative caching system for WMN [46]. Under this caching system, MAPs serve as user coordinators. Cache placements are made through cell-based and network-based schemes. This paper divided the cache system into two layers: the MAP layer and the mesh client layer, which means that both MAPs and mesh clients have cache size, which does not fit our scenario. The caching is also divided into two levels: network level and cell level. At the network level, the MAP makes a cache decision for each data item, such as whether a data item should be cached or not. At the cell level, the specific node (i.e., MAP) for caching the data item is determined. These can be adjusted to fit our scenario.

Zhao, Zhang, Cao, and Das also proposed a cooperative caching system for wireless P2P networks with the aim of determining the best place for caching data [49, 50]. This asymmetric cooperative cache method can lower the overhead for data copying and the end-to-end transfer delay.

Additionaly, there is a paper that also specifically defines caching message types, equations to compute caches costs, etc. in detail. Xu, Wu, Wu and Cao investigated a cooperative caching scheme for WMN with a focus on maintaining data consistency [47]. Based on a hierarchical architecture, it

combines both push-based and pull-based approaches. The aim is to reduce communication costs and message losses in particular.

Under the Teaching Company Scheme (TCS), this research project will involve academic-industrial collaboration with P2 Mobile Technologies Limited (P2MT). P2MT is a Hong Kong-based wireless mesh network company, which is dedicated to designing and producing wireless mesh network devices, and offering wireless mesh networking solutions with the latest technologies. With its BackN technology (patent pending), the company provides high-speed wireless mesh networking solutions based on the IEEE 802.11n standard. This technology enables an IEEE 802.11n-based multi-hop wireless mesh network to provide 300 Mbps bandwidth across the whole network (i.e., end-to-end). Compared to traditional AP-Controllers and Repeater Models, P2MT products can provide unlimited multiple-hop wireless extensions without losing channel bandwidth.

The company is also engaged in wireless networking research. For example, P2MT has recently published a journal paper in the IEEE Transactions on Mobile Computing, which provides a solution to the multi-hop TCP throughput degradation problem in IEEE 802.11n WMNs due to the increase in TCP Round Trip Time (RTT) through the use of multiple parallel TCP connections [23]. Additionaly, P2MT received the Silver Award of the Hong Kong ICT Award 2012 - Best Ubiquitous Networking (Mobile Enterprise Solution). Today P2MT's wireless mesh networking products are widely adopted in commercial, educational, and other public environments (e.g. CUHK, MTR station, shopping malls, etc.) in Hong Kong, Taiwan, and Mainland China.

This study will first establish a cloud-based wireless mesh network with data storage and other network service functionalities by using mesh router equipment provided by P2MT, then develop an adaptive data management system on this cloud-based WMN.

2.2 Cloud Computing

In recent years, cloud computing has become one of the hottest topics in both the academic and industrial arenas. Cloud computing allows computing resources to be shared through an on-demand utilization mechanism, which provides more convenient, powerful, and cost-effective computing services.

In [3], Armbrust, Fox, Griffith etc. illustrated that cloud computing can achieve the long-term vision of making computing as a utility service. It is expected that the demand for cloud computing will grow dramatically and virtualization of resources will become increasingly important. According to the NIST definition of cloud computing [32], cloud computing is a utility-based model that provides a configurable resource pool (e.g., storage, application, servers) to be shared ubiquitously and conveniently. There are five essential characteristics for the cloud computing model:

On-demand self-service

Resources (e.g., server, network, storage, computing capacity) should be able to automatically self-adjust based on utility requirement.

Broad network access

Resources should be available via standard network mechanisms and heterogeneous client platforms (e.g., PC, workstation, mobile devices).

Resource pooling

Cloud should provide heterogeneous resource pools to satisfy demands from customers. Internal structure (e.g., location, implementation detail) of resources, regardless of whether it is a physical resource or a virtual resource, should be transparent to customers.

Rapid elasticity

Resources should be able to scale rapidly and elastically with demand. Customers should also have the ability to provision the resources at any level at any time.

Measured service

Cloud system should be capable of monitoring, reporting, managing, and optimizing resources through a usage-based mechanism (i.e., based on resource utilization).

In terms of service models, three types of cloud computing service models are widely used:

Software as a Service (SaaS)

Provide software applications to customers which can be accessible using various clients through a cloud infrastructure without exposing underlying

infrastructure and computing resources (e.g., network, storage, operating systems).

Platform as a Service (PaaS)

Provide running environment for hosting a customer's application on a cloud infrastructure. Customers cannot control underlying infrastructure details and resources (e.g., network, storage, operating systems), but only configuration settings of the application.

Infrastructure as a Service (IaaS)

Provide provisionable computing resources such as processing, storage, operating system etc. to customers for deploying and running applications with more controls. However, customers still cannot control the underlying infrastructure of the cloud.

For cloud deployment, four types of deployment models can be adopted:

Public cloud

The cloud infrastructure can be used by general public, including business, academic, government.

Private cloud

The cloud infrastructure can only be used by people of an organization (i.e., exclusive use).

Community cloud

The cloud infrastructure can only be used by people or organizations of a community with a common interest.

Hybrid cloud

The cloud infrastructure is a mixture of public, private and/or community clouds.

With the rapid growth and development of the Internet, in recent years conventional computing and storage devices (e.g., single PCs, servers, workstations, and mainframes) were becoming overwhelmed. Hence cloud computing began, becoming the dream solution for large scale software or services. In addition to rich academic research inspired by cloud computing, various leading technologies have also been developed by industry. According to [38], there are

three major cloud computing styles divided by resource abstraction techniques from leading technology companies:

Server Virtualization

Also referred as *Amazon style*. From 2006 to 2007, Amazon released Amazon Web Service (AWS), consisting of Xen-based Elastic Compute Cloud (EC2), object storage service (S3) and structure data storage service (SimpleDB). AWS adopts on-demand and a more cost-effective charging rate, which soon made Amazon the pioneer of IaaS provision.

Technique-specific sandbox

Also referred to as *Google style*. From 2003 to 2006, Google published several research papers on PaaS cloud computing, and in 2008 released Google App Engine (GAE), which works as a sandbox container for hosting guest applications.

Hybrid

Also referred to as *Microsoft style*. This style combines both virtualization and technique-specific sandbox techniques. Microsoft released Azure in 2008, which adopts Windows Azure Hypervisor (WAH) as the underlying cloud infrastructure and .NET framework as its application container.

Table 2.1: Major Cloud Styles in Industry and Adopters

Server Virtualization (Amazon Style)	**Technique-specific sandbox (Google Style)**	**Hybrid (Microsoft Style)**
Amazon EC2 (Xen)	GAE (Python & JVM)	Microsoft Azure
21vianet CloudEx (Xen)	Heroku (Ruby)	(WAH & .NET)
GoGrid (Xen)	Morph Application	
RackSpace Mosso (Xen)	Platform (Ruby)	
Joyent (Accelerator)		
AT&T Synaptic (Vmware)		
Verizon CaaS (Vmware)		

Table 2.1 shows representative adopters of these three major cloud implementation styles in the market.

For the cloud-based wireless mesh network and considering the use cases of P2MT WMN products, this data management system should involve functionalities for a file hosting service. Three types of file hosting services are

considered in this case: Personal file storage to allow mesh clients to store personal/temporary files on the cloud-based mesh network; software file hosting to store data for commercial promoting software/applications; and content caching to decrease repeated network costs due to access locality.

2.3 Distributed File System

Distributed file system (DFS) has been a popular research topic for decades, and also the most widely used mechanism for simultaneously storing/sharing data among multiple hosts/servers [40]. In recent years, DFS techniques have been widely employed by industry to support cloud computing services. Borthakur and Dhruba have proposed the architecture and design [5], then Shvachko, Kuang, Radia, and Chansler formally proposed The Hadoop Distributed File System (HDFS) [43], which has now become the most famous DFS solution, providing a DFS along with a framework for analyzing and transforming huge data sets by using the MapReduce [13] paradigm. In [45], Weil, Brandt, Miller, and Long presented Ceph, another popular DFS with high scalability and performance. Ghemawat, Gobioff, and Leung proposed The Google File System (GFS) [19], which focuses on large distributed data-intensive applications with high aggregate performance to a large number of clients. In [41], Kistler, Kumar, Okasaki, Siegel, and Steere presented Coda, which is a large file system designed for Unix terminals under a distributed computing environment. Other popular DFS solutions, including Gluster [20], Lustre [6], MooseFS [34], have also been widely used, and have inspired a great deal of related research.

Table 2.2: Summary & Comparison of Popular DFSs

	HDFS	**Ceph**	**GlusterFS**	**Lustre**
Architecture	Centralized	Distributed	Decentralized	Centralized
API	CLI, FUSE, REST	FUSE, REST, mount	FUSE, mount	FUSE
System Availability	No failover	High	High	Failover
Data Availability	Replication	Replication	RAID-like	No
Replication	Async.	Sync.	Sync.	RAID-like

[15] has comprehensively studied and analyzed several existing DFS solutions. Table 2.2 shows the most popular four, which are widely used in the industrial space. Dimakis, Godfrey, Wu, Wainwright, and Ramchandran have discussed applying network coding into distributed storage systems to optimize the trade-off between storage and repair bandwidth [16]. The common denominator between all of these DFS solutions is that they all feature powerful performance with a high data throughput rate. Some are very strong and stable with great strength, robustness, and availability. However, their requirement for high computing capacity makes it impossible to host these systems on low-capacity computing equipment, such as mesh routers. Hence, we need another lightweight solution to equip WMNs with DFS function.

3 Person-based Adaptive Data Management Mechanism

The aim of this chapter is to investigate adaptive data management mechanisms for a cloud-based WMN (i.e., how to store and manage data effectively and efficiently over a WMN to provide cloud computing-like services). This chapter is inspired by and/or based on caching mechanisms for WMNs (e.g., [12, 14, 46, 47]), routing issues (e.g., [17, 33]), scheduling methods (e.g., [8]), and architectural issues relating to WMNs (e.g., [30, 36, 42]).

3.1 System Design & Architecture

Fig. 3.1 shows the basic system architecture. Basically a wireless mesh network (WMN) consists of the following network components: mesh access points (MAPs), mesh clients and, possibly, network gateways. MAPs, also known as mesh routers, are responsible for communicating with mesh clients, and for routing and forwarding data over the WMN. Mesh clients are mobile devices for providing connections between users and the network. A mesh client usually connects to one MAP. When a mesh client moves from one MAP to another MAP, the connection should still be maintained. Network gateways are optional for a WMN. They provide connection to the Internet. To provide cloud- based services, each MAP can provide storage capability. The storage functionality can be implemented either through a general Linux virtual file system (VFS) on mesh routers, or through a lightweight distributed file system (e.g. GlusterFS, Ceph, etc.) to form a virtual file system on the cloud. Furthermore, MAPs can communicate with each other by conveying messages for data management purposes. Note that data can be forwarded and replicated over the network for adaptive data management purposes. Mesh clients can upload and download data using an effective mechanism.

In addition to network communications service, the proposed cloud-based WMN can also support uploading/downloading functions (e.g., the uploading and downloading of personal data files to/from a WMN). Data can be stored in a MAP under each user account with proper access rights. For example, a user can store frequently used data files in a cloud-based WMN. As users can

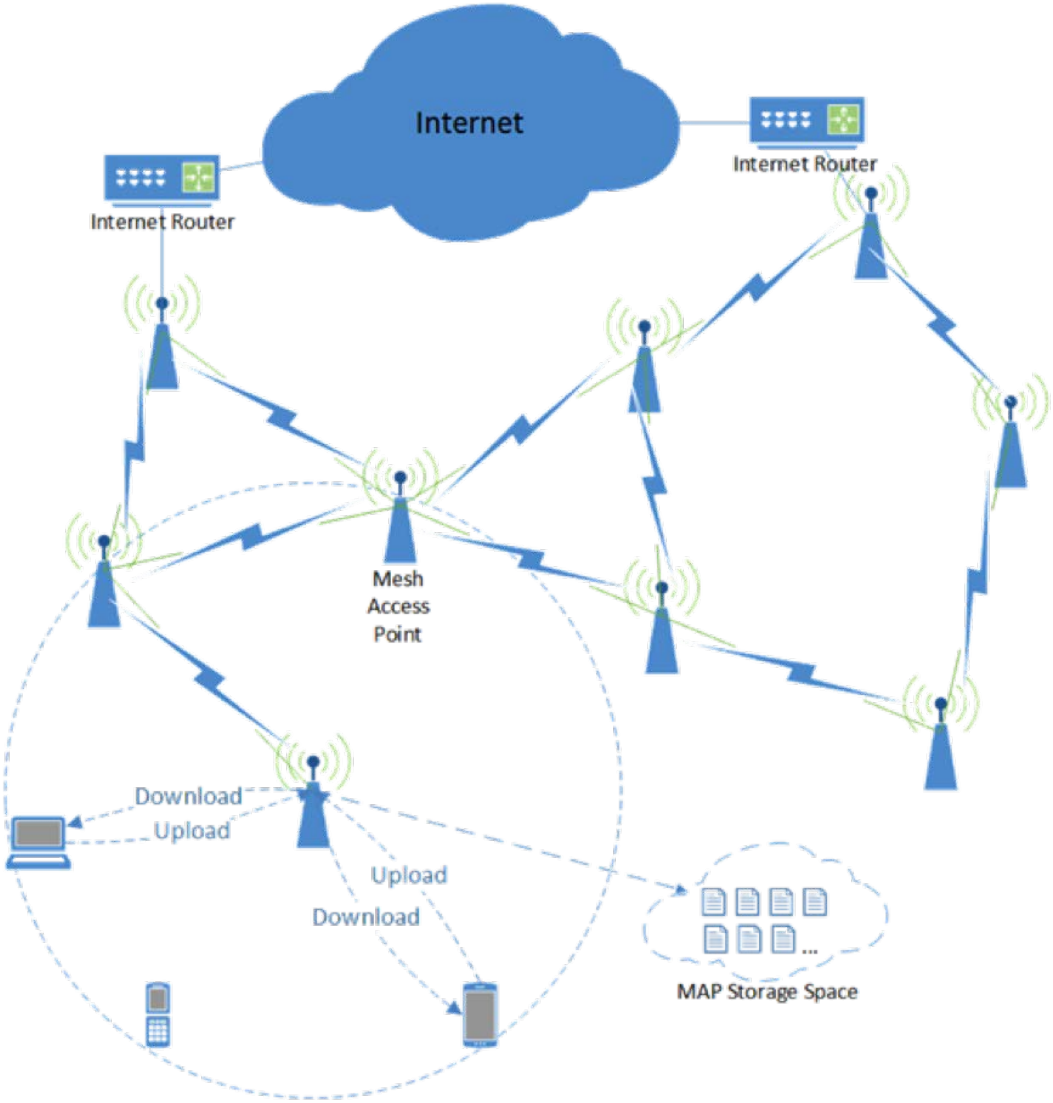

Fig. 3.1: Architecture of a Cloud-based Wireless Mesh Network

move around, the WMN should keep track of the users' locations so that data can be moved based on user locations (i.e., certain data can follow a user). Basically, when a user moves from one MAP to another, some data files can be moved from the previous MAP to the current MAP, depending on certain requirements. This mechanism seeks to enhance the access or retrieval time for the user data (e.g., important files).

However, moving data frequently, especially large data files, can be costly in terms of the consumption of network resources. Therefore it should be desirable to move certain data files only, taking into consideration various factors, such as user mobility, MAP storage, access probabilities, and so forth.

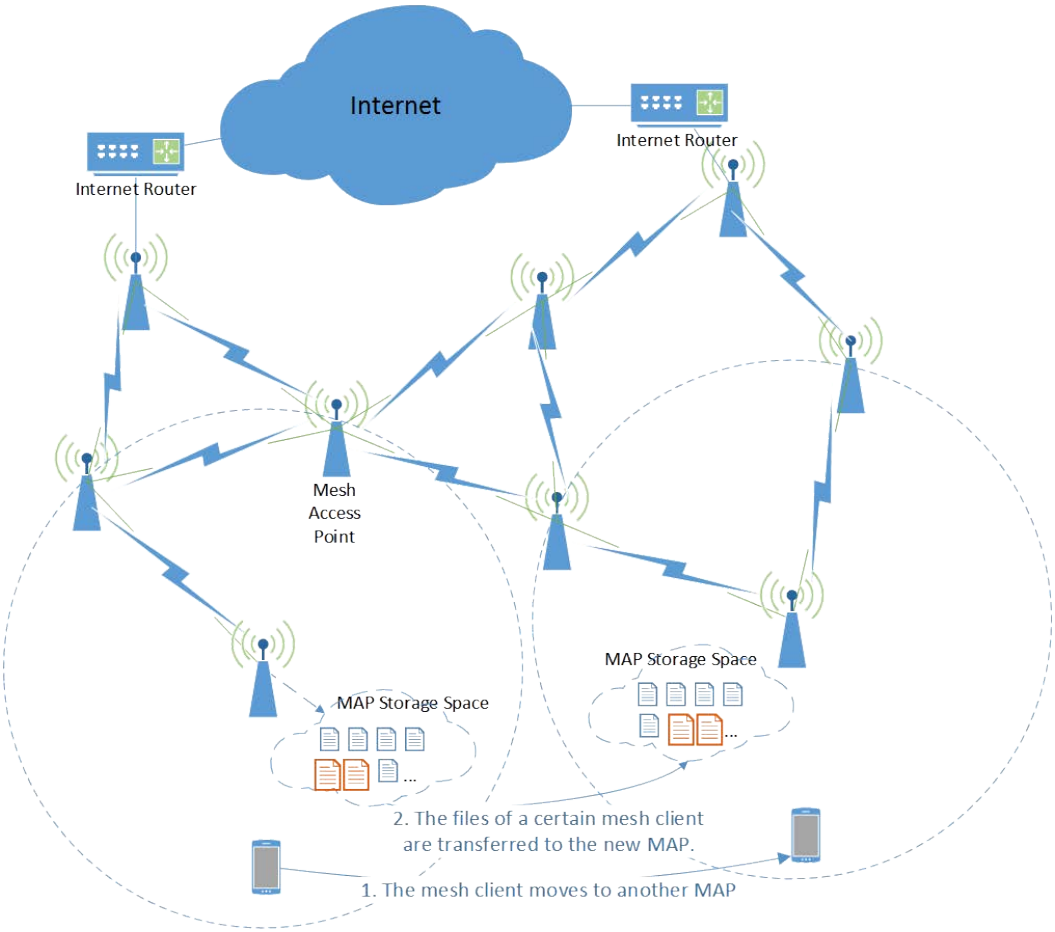

Fig. 3.2: Person-based Adaptive Data Management Mechanism

3.2 Hardware & Prototype

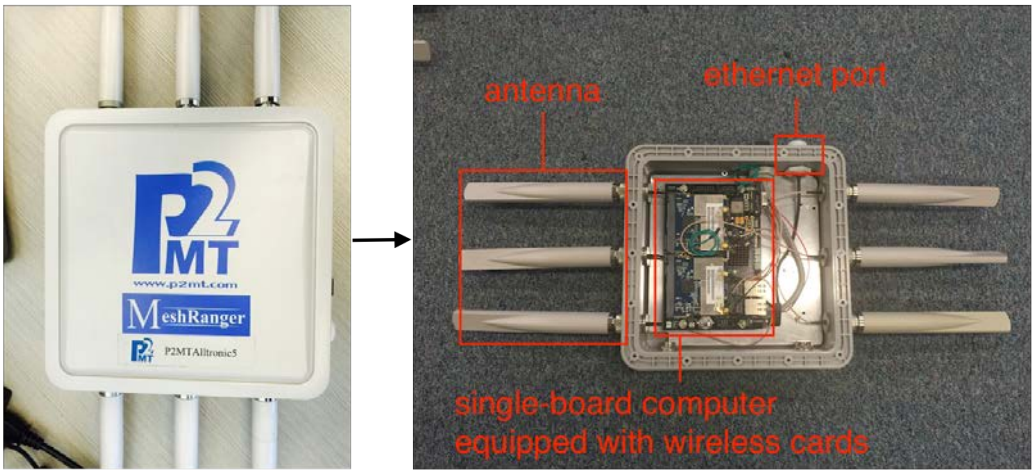

Fig. 3.3: Router model MeshRanger from P2MT

A prototype based has been developed on WMN routers provided by P2MT (Fig. 3.3). Each WMN router consists of three major components: one mother

board, up to four wireless network interface controllers, and antennae. The motherboard is a single-board computer equipped with a 600MHz dual core CPU, 256Mbytes of SDRAM memory, and 16Mbytes of flash system memory [31]. It has four mini-PCI sockets, which can support up to four wireless network interface controllers (either 802.11n 2.4GHz or 5GHz, or dual-band) to implement mesh router functionalities, and two Ethernet Ports for wired connections (for configuration or integration with other wired networks). There are also two RS232 serial ports, a micro SD flash expansion socket, and USB ports for expansion and other purposes.

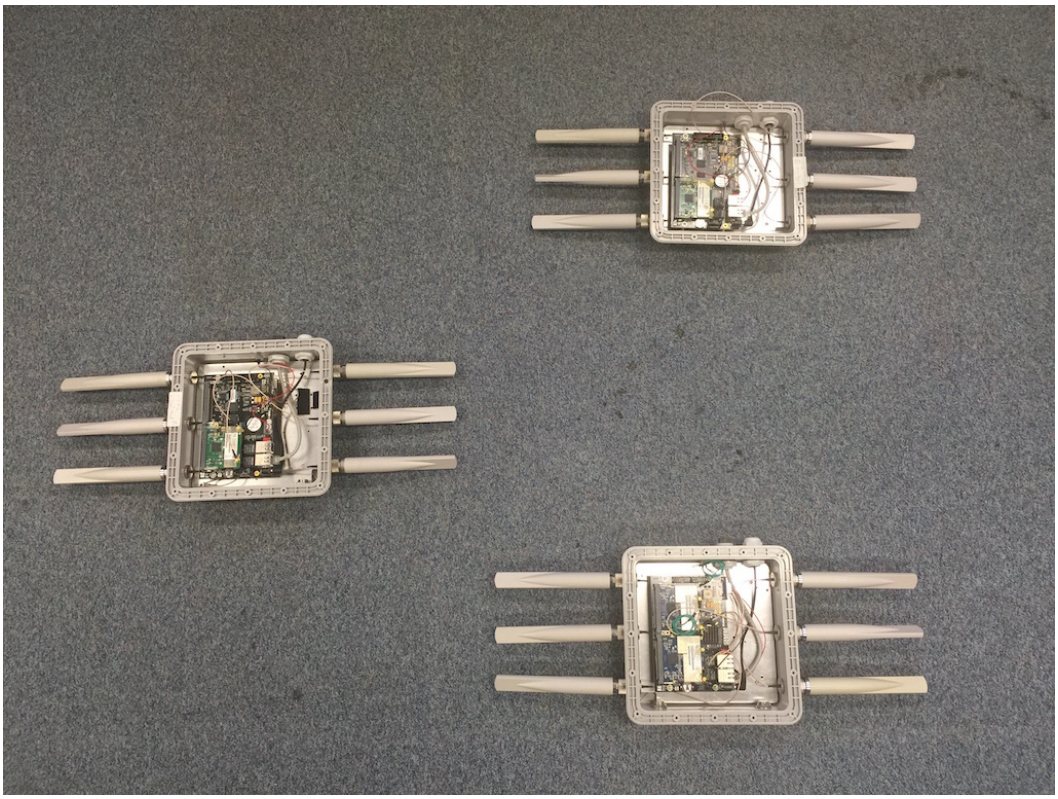

Fig. 3.4: Wireless Mesh Network Formed by MeshRangers

For the operating system on WMN routers, a customized version of OpenWrt for the Laguna family (cns3xxx) is used. OpenWrt is a lightweight Linux distribution for embedded devices [18]. It is free, open-source, and community driven. Currently, OpenWrt is widely used by industrial vendors to facilitate product development and is also used by customers to customize device firmware in order to add more functions or enhance performance.

OpenWrt provides package management functions so that any packages can be chosen to suit different requirements and development purposes [18]. Programs are cross-compiled through SDK or Toolchains built from the Laguna OpenWrt source together with all other necessary libraries needed for development. With

the package management features of OpenWrt, additional functionalities, such as routing management, a web server with a common gateway interface (CGI), databases, and so on, can easily be added to expand and customize the mesh routers.

Every WMN router used to build the wireless mesh network is loaded with at least one 2.4GHz and two 5GHz wireless network interface controllers. The 2.4GHz interface is used to access wireless networks for mesh clients, and 5GHz interfaces are used to build the backbone of the mesh network.

There are three ways to establish interconnections among mesh routers: Access Points (AP) to Station (STA) mode, Ad-Hoc mode, and Wireless Distribution System (WDS) mode. In this study, the AP-STA mode is adopted, where the backbone interfaces (the two 5GHz ones) act as either AP or STA, and are connected as pairs. Because different subnet IP addresses are assigned to these paired connections, the Optimized Link State Routing (OLSR) protocol is used to rout packets among these interfaces to make different IP subnets accessible. Fig. 3.5 shows one of the possible topological configurations for the wireless mesh network. Five mesh routers are connected by four links in four different subnets (192.168.1-4.0/24). For mesh routers to be in two subnets simultaneously, two wireless network interfaces are used as mentioned above, and the routing tables are built with the tool orsld (OLSR daemon).

3.3 File System

3.3.1 Storage Function with Linux File System

In the prototype, the normal Linux file system is used for the purpose of storing data. All data are stored in a particular directory on the mesh router. This section focuses on describing how the data of mesh clients are stored on the entire mesh network (i.e., in a distributed manner), and how the data management system can locate particular data. Since every mesh router in the mesh network is capable of storing data for mesh clients, the data management system needs to know where a particular file is stored. One method of locating it is to keep the index of the data stored so that the data management system can make a direct search of the index to obtain the location of the data. A database is used for maintaining these indices. Fig. 3.6 is a Unified Modeling Language (UML) diagram of the database to illustrate the schema.

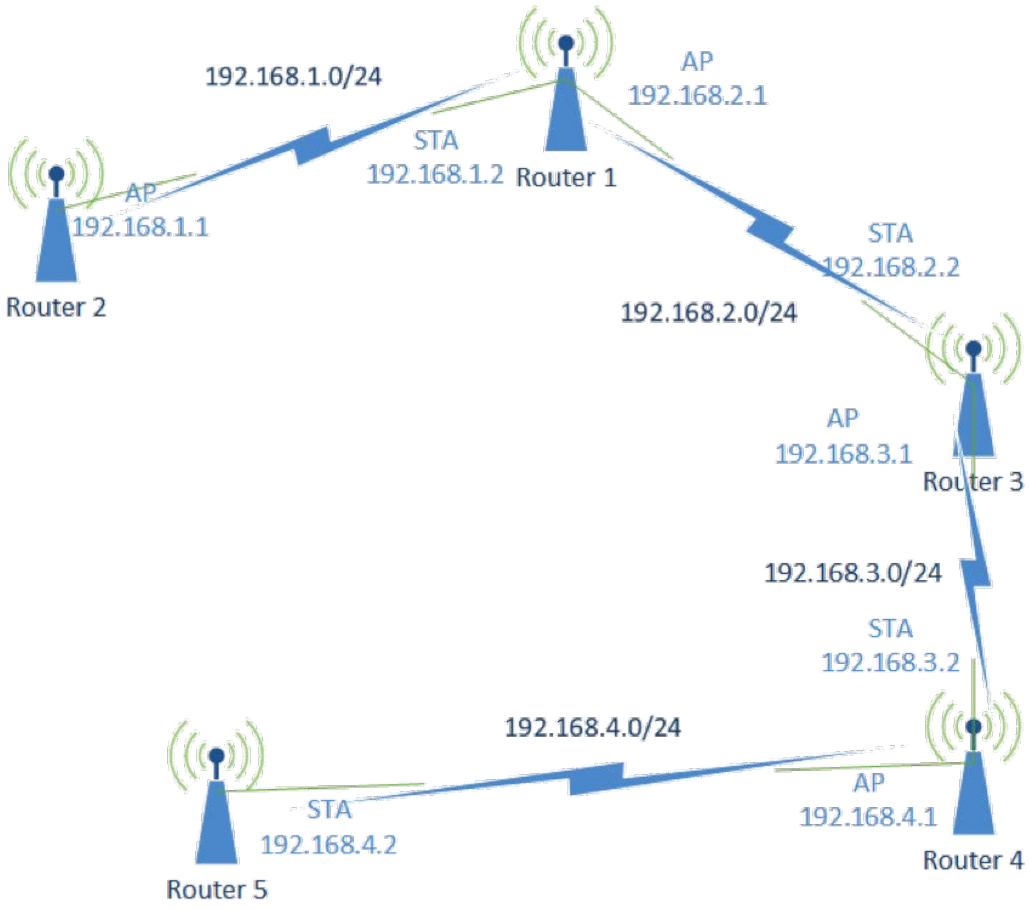

Fig. 3.5: One of the Possible Topologies of the WMN Backbone

There are three types of main entities in the schema: *Client*, *File*, and *Router*. A unique ID is assigned to each entity to maintain its uniqueness and prevent unnecessary redundancies. The schema of this database follows the principles of "who, what, and where". We combine these three kinds of unique ID under a fourth table named *FileStorage*, which is the index that records which file (what) of the mesh client (who) is stored on which router (where).

Table *Client* records the basic information of mesh clients, such as the hostname, MAC address, and time of creation. Information such as the time of the last connection and the number of logins is also recorded for other management purposes. A mesh client may own multiple file storage indices because a client can own multiple files, and these files may also be duplicated on different routers. However, a file storage index can only belong to one client because it is uniquely identified by a foreign key *clientID*.

Table *File* records the basic information of a file, such as its name, size, and time of creation. Information such as the time of the last access and

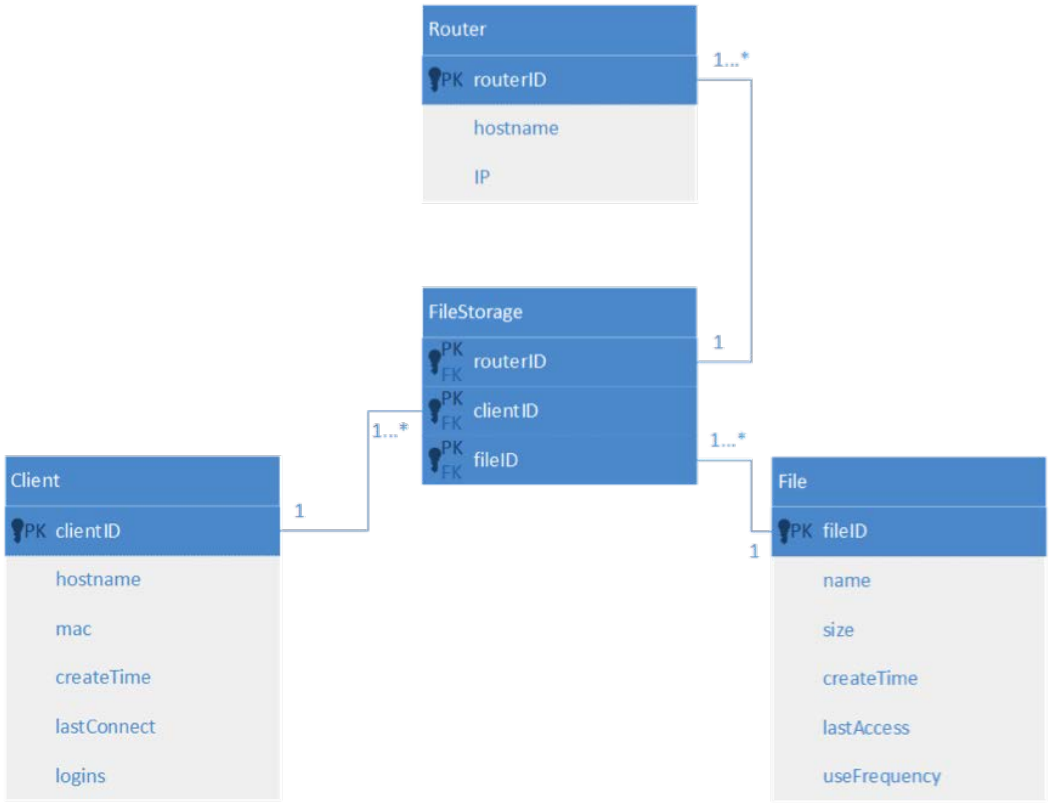

Fig. 3.6: UML Diagram for Data Location Index

```
mysql> desc Client;
+-------------+-----------+------+-----+-------------------+-----------------------------+
| Field       | Type      | Null | Key | Default           | Extra                       |
+-------------+-----------+------+-----+-------------------+-----------------------------+
| hostname    | text      | NO   |     | NULL              |                             |
| mac         | char(18)  | NO   |     | NULL              |                             |
| clientID    | char(64)  | NO   | PRI | NULL              |                             |
| createTime  | datetime  | NO   |     | NULL              |                             |
| lastConnect | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| logins      | int(11)   | NO   |     | 0                 |                             |
+-------------+-----------+------+-----+-------------------+-----------------------------+
```

Fig. 3.7: Schema of Table Client in MySQL

frequency of use is also recorded for other management purposes. Similar to the relationship between mesh clients and the file storage index, a file may also own multiple file storage indices because a file can be shared by multiple mesh clients, or be duplicated on different routers. A file storage index only belongs to one file because it is uniquely identified by a foreign key *fileID*.

Table *Router* records the basic information of a mesh router, such as its hostname and IP address. A mesh router may own multiple file storage indices because it can accommodate multiple files in storage. However, a file storage index only belongs to one mesh router because it is uniquely identified by a foreign key *routerID*.

```
mysql> desc File;
+--------------+------------+------+-----+-------------------+-----------------------------+
| Field        | Type       | Null | Key | Default           | Extra                       |
+--------------+------------+------+-----+-------------------+-----------------------------+
| name         | text       | NO   |     | NULL              |                             |
| fileID       | char(64)   | NO   | PRI | NULL              |                             |
| size         | bigint(20) | NO   |     | NULL              |                             |
| createTime   | datetime   | NO   |     | NULL              |                             |
| lastAccess   | timestamp  | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| useFrequency | int(11)    | NO   |     | 0                 |                             |
+--------------+------------+------+-----+-------------------+-----------------------------+
```

Fig. 3.8: Schema of Table File in MySQL

```
mysql> desc Router;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| hostname | text     | NO   |     | NULL    |       |
| ip       | char(16) | NO   |     | NULL    |       |
| routerID | char(64) | NO   | PRI | NULL    |       |
+----------+----------+------+-----+---------+-------+
```

Fig. 3.9: Schema of Table Router in MySQL

The file storage index links mesh clients, files, and mesh routers together and uniquely identifies the ownership and location information. Hence, from any side of this triangular relationship, the other two sides can easily be accessed when an enquiry for information is made.

```
mysql> desc FileStorage;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| clientID | char(64) | NO   | PRI | NULL    |       |
| fileID   | char(64) | NO   | PRI | NULL    |       |
| routerID | char(64) | NO   | PRI | NULL    |       |
+----------+----------+------+-----+---------+-------+
```

Fig. 3.10: Schema of Table FileStorage in MySQL

3.3.2 Storage Function with A Distributed File System

Basic requirements can be met by using a Linux file system and by ensuring that the system can operate properly in an experimental environment; however, some problems may be encountered in the practical use of the system. A critical problem is the Single Point of Failure (SPoF). For example, if one of

the mesh routers encounters network problems or experiences system failure, then the data of the mesh clients that were stored on this router can no longer be accessed; likewise, if the mesh router hosting the file storage index database encounters network problems or experiences system failure, then the entire data management system will fail. To avoid such fatal situations, it is necessary to have a replicate copy of the data and file storage indices of the mesh clients. This will improve the reliability, accessibility, and fault tolerance of the data management system.

Adopting a distributed file system is a good solution for such requirements. Two important features of a distributed file system are access transparency and replication transparency, which means that what mesh clients see in the distributed file system is exactly the same as what they see in a local file system, so they will not regard the replications made by the distributed file system as fail-safes.

Currently, there are various distributed file systems. They are designed for different goals, different system scales, and different considerations. In the next chapter, the use of a distributed file system will be studied for an effective adaptive data management system.

3.4 Implementation

In this section, I present a person-based adaptive data management mechanism, which provides upload/download functions for mesh clients, and adaptively moves files along with the movements of the owner to enhance the access speed. Hence, the development of this section has been divided into two parts: (i) a web User Interface (UI) to let mesh clients upload and manage personal files; (ii) an agent program running on mesh routers to monitor the mobility of mesh clients, and to adaptively decide whether to move the files and on which mesh router the files should be placed.

3.4.1 Web UI for File Uploading/Management

The web UI is responsible for handling file uploading/ management operations, similar to current commercially available online drive/sync services. The following is a brief description of some of the ways in which the web UI can be used, as examined in this study:

1. A mesh client accesses the web UI, and the web UI prompts the client by asking the client to log in, or to go directly to the main page if the client is already logged in;

2. The mesh client logs in through an account/device, and the web UI displays the main page containing all of the files belonging to the logged on mesh client;

3. The mesh client uploads new file(s), and the web UI stores the uploaded files in file system, inserts information on the file, and builds a file storage index in the database;

4. The mesh client does management operations (e.g., renames, deletes, downloads, etc.) on existing file(s); the web UI responds accordingly, and updates/deletes the file information/file storage index to/from the database.

Note that the web UI does not need to be concerned about which type of storage file system is used (the normal Linux file system or the distributed file system) because they are separate. If the normal Linux file system is used, the placement of the file will be handled by the agent program, which will be described in the next section; if the altered distributed file system is used, the storage location will be determined by the altered distributed file system.

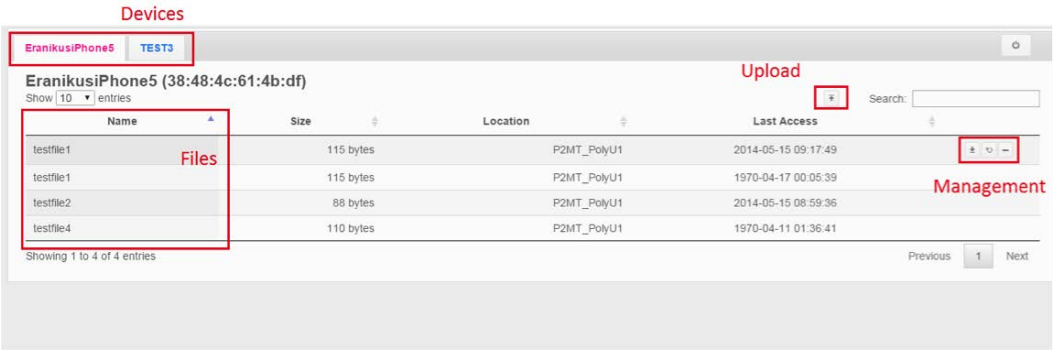

Fig. 3.11: Web UI for Person-based Adaptive Data Management System

The front end of this web UI is developed using the JQuery library with multiple plugins, such as the JQuery-UI, JQuery DataTables, JQuery Download, and JQuery Fileupload. The back end of the web UI is written using PHP, and an UploadHandler plugin is adopted to achieve the drag-and-upload function. MySQL is used as the database to keep the file information and file storage indices. The web UI is deployed on one of the mesh routers, normally the

network gateway of the wireless mesh network. The index database is placed on another mesh router that, for reasons of security, cannot be directly accessed from an external network environment.

3.4.2 Agent Program for Monitoring the Mobility of Clients and Making Decisions on the Movement of Files

The agent program runs on all mesh routers with storage functionality inside the mesh network. This agent program is responsible for three major tasks: monitoring the movements of mesh clients, making decisions on the movement of files, and transferring files to a target router if required. Instead of having a central router conduct all computations, agent programs running on all mesh routers work in a decentralized way to share the work load. Each mesh router retrieves and computes information individually, and exchanges information with fellow routers if needed. Communications between routers are carried out through UDP messaging. The agent program is written in C language, with assistance from shell scripts and external libraries (e.g., libmysqlclient, libssh2, libgcrypt, etc.).

In order to make files follow along with a mesh client's movements, the data management system needs to be aware of the mobility behaviour of the mesh client. A mesh router should make a file movement decision when a handoff operation occurs. Hence, the agent needs to monitor connections from mesh clients, and triggers a decision on the movement of a file when a new connection is built. Since OpenWrt is a Linux distribution for embedded devices, especially for network routers, shell scripts can be used to help retrieve information on connected devices instead of collecting information from scratch. The following piece of shell script code provided by OpenWrt [18] can return the IP address, hostname, and MAC address of the current connected devices:

```
1  for interface in `iw dev | grep Interface | cut -f 2 -s -d' '`
2  do
3      # for each interface, get mac addresses of connected stations/clients
4      maclist=`iw dev $interface station dump | grep Station | cut -f 2 -s -d' '`
5
6      # for each mac address in that list ...
7      for mac in $maclist
8      do
9          # If a DHCP lease has been given out by dnsmasq, save it.
10         ip='UNKN'
```

```
11          host=''
12          ip='cat /tmp/dhcp.leases | cut -f 2,3,4 -s -d' ' | grep $mac | cut -f 2 -s -d'
             '`
13          host='cat /tmp/dhcp.leases | cut -f 2,3,4 -s -d' ' | grep $mac | cut -f 3 -s -
             d' ''
14
15          # Show the mac address:
16          echo $ip,$host,$mac
17      done
18  done
```

The agent program runs this piece of shell script periodically to retrieve information on currently connected mesh clients, and compare this information to the past client list retained from the last timeslot. If the current client list contains a new member that does not appear in the past client list, then this client is a newly connected mesh client, and the process of making a file movement decision should be triggered. If a member from the past client list does not appear on the new client list, this means that this mesh client has disconnected from this mesh router. The client may have been disconnected from the wireless mesh network, or have moved to another mesh router.

The following is the pseudo-code of the algorithm, which monitors mesh client movement. The algorithm is run periodically:

The decision to move a file is made by calculating a score called the file transfer priority. During the process of making a decision to transfer a file, the file transfer priority value of all of the files belonging to a mesh client will be calculated and sorted in order to determine whether or not to transfer a file. As an example, we consider that the file transfer priority p is related to:

$$f = use\ frequency$$

$$s = file\ size$$

$$t = time\ of\ last\ access$$

For this illustrative example, we assume that the priority value p is directly proportional to frequency f, and inversely proportional to file size s and time to last access t:

$$p \propto \frac{f}{st}$$

Algorithm 1 Monitor Movements of Mesh Clients

Input: *pastClientList*
Output: *pastClientList*

```
1:  procedure MoniterMovement
2:      newClientList ← GetClientList shell script
3:
4:      for each client in pastClientList do
5:          // do nothing to clients that not moving
6:          if client exists in newClientList then
7:              remove client from newClientList
8:          end if
9:
10:         // remove clients if moved out
11:         if client does not exists in newClientList then
12:             call clientDisconnected(client)
13:             remove client from pastClientList
14:         end if
15:     end for
16:
17:     // the rest clients in newClientList are newly connected clients
18:     for each client in newClientList do
19:         add client to pastClientList
20:         // do file movement decision making
21:         call clientConnected(client)
22:     end for
23: end procedure
```

However, the range of file size f and time to last access t may have large values, so they need to be normalized within a reasonable range. According to the properties of the logarithm function, the log value will increase much more slowly when the actual value becomes larger. Therefore, we use the logarithm function for normalization purposes.

Since the file size is represented by bytes, the value of $\log_2 s$ will be 10, 20, 30, and 40 for 1KB (2^{10} bytes), 1MB (2^{20} bytes), 1GB (2^{30} bytes), and 1TB (2^{40} bytes), respectively. A UNIX timestamp is used to represent the time to the last access t, with t being equal to the current timestamp minus the last access timestamp. We can also use a logarithm to control the range of value t (e.g., $\log_2 t$ will be around 16, 21, or 25, if the file has not been accessed for one day, one month, or one year, respectively);

Now the equation becomes:

$$p = \frac{f}{\log_2 st}$$

In most situations, the value of $\log_2 st$ is less than 100, so the scale is much more acceptable for a priority value. Note that the above is just an example - other policies can also be used.

After calculating the priority values of all of the files, the agent program will decide whether or not to transfer files to the target mesh router. Other information that may be involved in this decision-making process (e.g., the possible transfer cost, the traffic of the mesh network, etc.) will be retrieved through messaging communications between mesh routers.

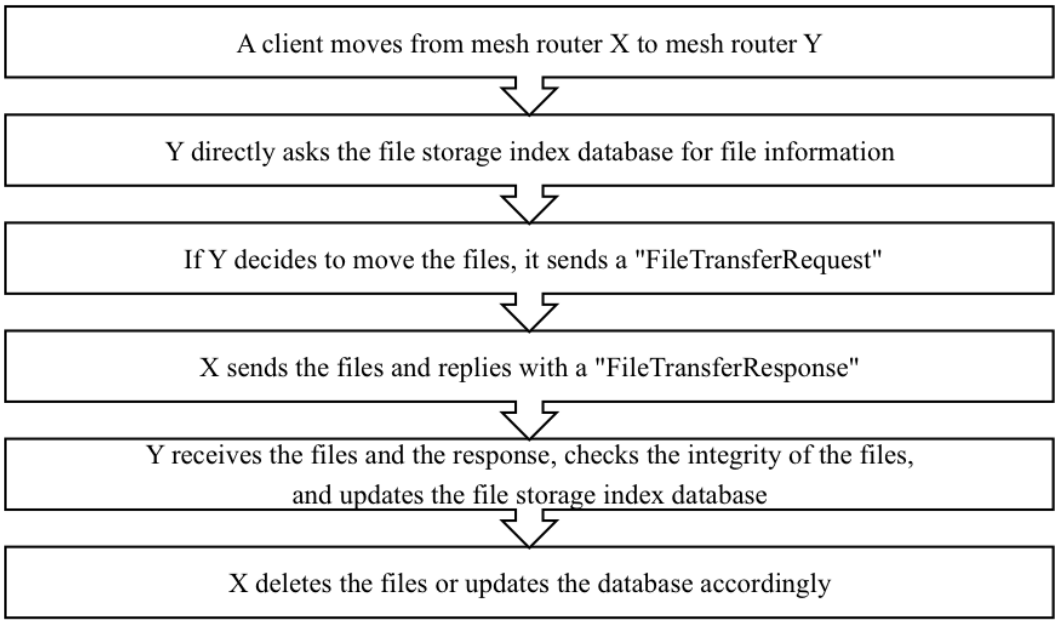

Fig. 3.12: Work Flow of the File Transfer Operation

If the agent program decides to transfer a particular file from another mesh router to itself, the file transfer operation will be performed. The method of transferring a file depends on the type of storage system. If a normal Linux file system is used for storage functionality, the current mesh router will send a $FileTransferRequest$ to the mesh router that holds the desired file. After the target router receives the request, the file will be sent directly to the requesting mesh router through Secure Copy (SCP) with a $FileTransferResponse$. Next, the requesting mesh router receives the file and the response, checks the integrity of the received file, updates information in the file storage index database, and sends an acknowledgement to the sender with a $FileTransferSucceed$. The sender will decide whether to remove the file from its local storage or to keep it, depending on whether there are other ownership relations existing between other mesh clients and this file. If the file is to be kept, the respective

information in the file storage index database will be updated accordingly. Fig. 3.12 shows the work flow of the file transfer operation.

3.5 Conclusion

In this chapter, I have presented a cloud-based wireless mesh network with cloud storage functionality, with an accessible web UI for uploading/managing files, and an agent program to monitor the movements of mesh clients and adaptively move files among mesh routers to reduce access costs. The person-based adaptive data management mechanism can improve data access efficiency for mesh clients. In the following chapters, apart from further developing the person-based data management mechanism, a distributed file system specifically designed for cloud-based wireless mesh network will be discussed, and a group-based data management mechanism, which considers data to be shared by multiple mesh clients, will be investigated.

4 MeshFS: A Distributed File System for Cloud-based Wireless Mesh Network

To better achieve requirement of person-based and group-based adaptive data management mechanisms for cloud-based wireless mesh networks, and inspired by cloud computing and DFS techniques, MeshFS, a distributed file system specifically designed for cloud-based WMN is presented in this chapter. MeshFS integrates scattered storage resources from mesh routers to provide a mountable file system interface to Unix/Linux file systems with a fault-tolerant feature, allowing limited resources from mesh routers to be better utilized.

4.1 Objectives & Features

For most existing DFS solutions, the objectives and features mainly focus on providing high data throughput and the capacity for handling huge amounts of data. Normally these kinds of DFSs will be deployed on powerful servers with a high volume of storage disks. Different from other DFSs, the DFS to be adopted in this study will be accommodated by mesh routers (introduced in Section 3.2), which are single-board computers with very limited computing capacity and storage space. This hardware limitation became the first reason to develop a new DFS solution specifically designed for mesh routers with low computing capability.

Second, the mesh routers used in this study are running OpenWrt as firmware and operating system. Although OpenWrt is essentially one of the many distributions of the Linux family, it still has many derivations from other desktop or server versions of Linux systems. At the start, we tried to transplant and compile GlusterFS onto OpenWrt, but after many trials, the results indicate a dead end due to the lack of libraries required by the implementation.

Therefore, due to the hardware limitations and constraints on the firmware and operating system, we decided to develop our own light-weight DFS, which is specifically designed for mesh routers running OpenWrt firmware, named

MeshFS. The objectives and features of MeshFS differ from those of traditional DFSs. The main design goal of MeshFS is to integrate limited computing resources and scattered storage resources from the WMN to provide one general file system interface, which can be directly used like a normal local file system for applications on upper-layers. The following are the objectives that MeshFS is designed for and the features provided:

Transparency

MeshFS shall provide a compatible interface that allows clients to use it like a local Linux virtual file system. Clients shall be able to perform tasks/actions by calling Linux system calls on files and MeshFS itself, just as if the task/action were on a local file system (e.g., mount/unmount, change mode/permission, mknod/mkdir, read/write, etc.). The internal logic and procedures of MeshFS shall be completely invisible to clients.

Fail-safe

MeshFS shall provide continuous and robust services. In case of system failures causing data inaccessibility (e.g., file loss/corruption, mesh router crash, etc.), MeshFS should provide data backup and recovery mechanisms to prevent Single Point of Failure (SPOF) problems. These copies/replications should be consistent and synchronized. Once data failure occurs, MeshFS should be able to automatically recover the data from other copies/replications.

Flexibility

MeshFS should be able to adjust and fit due to scale change in the WMN. When a new mesh router is added into or an existing mesh router fails/is removed from the network, MeshFS should be scalable and able to operate according to the new topology of the network. MeshFS should also be modularized for better extensibility. Functional interfaces should be left for future upgrades and maintenance (e.g., by applying a new strategy for backup & recovery functionalities).

Simplicity

As a file system running on a lightweight embedded system with limited computing ability, MeshFS should not occupy much in terms of computing resources and network bandwidth. The number of operations and protocol messages exchanged during communications should be minimized. The design and structure of MeshFS should be simplified to provide stable, fast file services.

4.2 System Design & Architecture

In this chapter, the development work was conducted on the same hardware equipment and mesh network topology used in a previous stage of research on cloud-based wireless mesh networks and person-based adaptive data management mechanisms [48]. The mesh routers used for constructing the WMN and running MeshFS are single-board computers equipped with a 600MHz dual core CPU, 256Mbytes of SDRAM memory, and 16Mbytes of flash system memory [31]. The operating system running on mesh routers is a customized version of OpenWrt, which is a light-weight Linux distribution for embedded devices [18].

In MeshFS, mesh routers can perform as three different kinds of characters: Client, Server, and Metaserver. Mesh routers should be in the same local network or at least reachable to on another, so that they can communicate through messages. To perform as a character, a mesh router needs to run the respective agent program. As illustrated in Fig. 4.1, a mesh router can simultaneously behave as more than one character, and each character has different responsibilities:

Client

The Client program acts as the entry interface to access content in MeshFS for applications on upper layers. Through the Client program, MeshFS can be mounted to any directory in OpenWrt Linux, just like mounting a flash disk or any other kind of storage equipment. After MeshFS is mounted, the operating system and other applications can use it just as if it were a regular local file

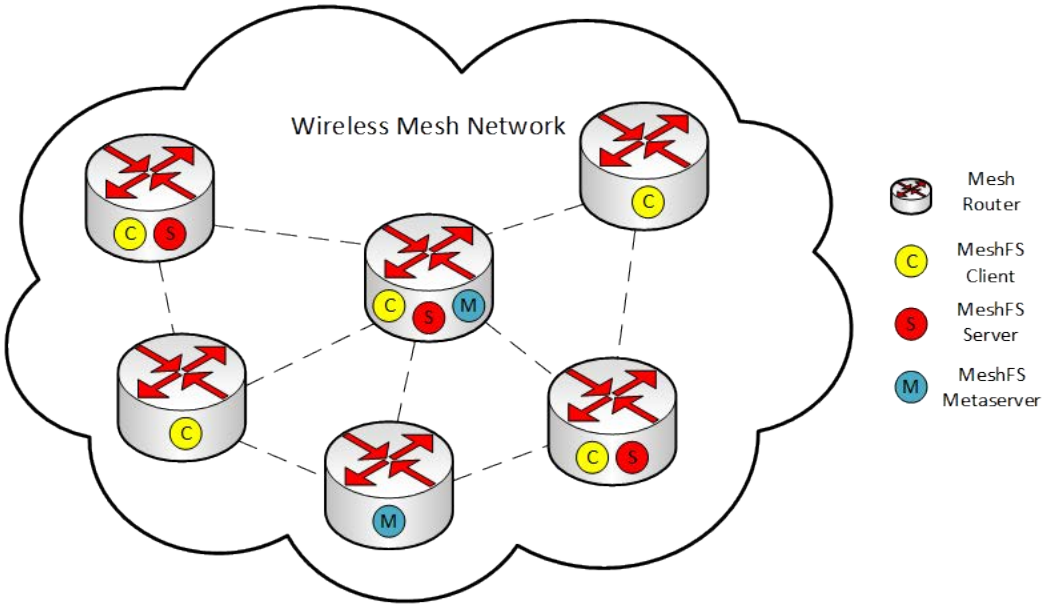

Fig. 4.1: Sample Architecture of MeshFS

system. Mesh routers that run Client to mount MeshFS can access/edit data in this MeshFS network.

The Client program needs to be assigned to one of the MeshFS Servers in the configuration - this could be the router itself if it is also behaving as a Server, or the nearest router that is acting as a Server. The Client program will communicate with the assigned Server in order to access/edit data in this MeshFS network.

Server

Mesh routers running the Server program will dedicate their disk spaces as storage nodes in this MeshFS network. The MeshFS Server will register itself to one Metaserver, hence, the MeshFS network and its members can become aware of the existence of the Server.

The Server program, which is a daemon run on the mesh router, will correspondingly listen for and handle incoming messages from the Client and Metaserver. For Clients, most requests are related to Linux system calls (syscall). For example, if a user types *ls* in a directory to which MeshFS is mounted, the Client program will send a *getattr* syscall request to its assigned Server. In response, the Server will retrieve all node attributes from the Metaserver and send them back to the Client. Sometimes, the Metaserver might send

operation-related requests to the Server(s) (e.g., making backups, transferring data, checking the integrity of the data, etc.); the Server then handles the requests using the respective implementation procedures.

Metaserver

The first responsibility of a Mesh Metaserver is to store the metadata of the MeshFS network. The following data will be stored in a MeshFS Metaserver: the metadata of files and directories (e.g., name, size, mode, etc.), the registered information of the Servers (e.g., hostname, IP address, etc.), the hierarchical structure of the file system, the physical location for storing files or backups, and so on. Unlike Clients and Servers, only a few Metaservers are needed for load balancing and preventing SPOF in a MeshFS network. If multiple Metaservers are adopted, all of this kind of information should be consistent and synchronized.

Besides storing metadata, Metaservers are also responsible for making and verifying backups for fail-safe service. A Metaserver daemon will perform the following procedures periodically: generating backups for files, checking the validity and integrity of existing backups, re-backup if a backup is invalid or a file is updated, and recovering data if a file is compromised.

4.3 Implementation

As mentioned in the previous section, there are three types of characters that mesh routers can perform in a MeshFS network. Hence, three agent programs are implemented to conduct the tasks of each one of them. All of the three agent programs are mainly implemented in C programming language, with assistance from the use of Shell scripts, SQL, etc.

4.3.1 Messaging

In order to work as a distributed system, messages need to be exchanged for inter-communications between the mesh routers. Communicating through messaging occurs frequently during the working time of MeshFS. The message payload may carry various types of information, such as command type, file metadata, file location, file content, and so on. Hence, a strong and

stable message exchanging protocol is required. To facilitate this part of the development, ZeroMQ (also known as ØMQ) is adopted as the messaging module.

ZeroMQ provides tools for developing parallel or distributed applications [22]. ZeroMQ allows users to set up message queues and sockets that are similar to the IP-based and Unix-based sockets but with high scalability, thus allowing the user to focus on handling messages instead of managing sockets. ZeroMQ supports four basic types of messaging: *Request-reply*, *Publish-subscribe*, *Push-pull*, and *Exclusive-pair*. In the development of our software, we chose the pattern of Request-reply to handle most messages.

To comply with the *Request-reply* pattern, the first message that is generated to start a procedure for performing a task is called a request, while further messages in the same scope of procedure are considered replies or responses. In MeshFS, all three kinds of agent programs may generate and send requests; however, only the Server module can receive and handles messages accordingly. The following is the general format of a request message:

$$< request >:< attr1 >, < attr2 > ...$$

A message starts with the request command and a colon, followed a sequence of attributes separated by commas, which are necessary to complete the request. Messages are sent out through ZeroMQ sockets. After receiving the message, the Server program will extract information from the message, then perform the logics accordingly. Details of the implementation for processing messages will be discussed in the next sections.

Using detailed formats, table 4.1 explains all of the requests that appear in MeshFS. The purpose of most requests is to perform the implementation logic for a specified Linux syscall; thus, they are given the same name as the syscall. Those requests prefixed by *op* indicate that these requests function for operational purposes, such as backup, recovery, and so on. A detailed explanation of the purpose of the requests will be provided later.

4.3.2 Client

As introduced earlier, one of the main responsibilities of the MeshFS Client program is to provide a compatible file system interface to the operating system.

Table 4.1: List of Request Messages

Request	Format
getattr	getattr:<path>
readdir	readdir:<path>
open	open:<path>
read	read:<offset>,<size>,<path>
write	write:<offset>,<size>,<path>
mknod	mknod:<path>,<mode>
unlink	unlink:<path>
mkdir	mkdir:<path>,<mode>
rmdir	rmdir:<path>
chmod	chmod:<path>,<mode>
rename	rename:<oldname>,<newname>
renameRemote	renameRemote:<oldNodeID>,<newNodeID>
utime	utime:<path>,<atime>,<mtime>
truncate	truncate:<path>,<size>
op_copy	op_copy:<sourceIP>,<nodeID>,<targetIP>
op_copy_request	op_copy_request:<nodeID>,<IP>,<path>,<port>
op_checksum	op_checksum:<type>,<nodeID>

Filesystem in Userspace (FUSE, also known as libFUSE) is a library that allows developers to implement a fully functional file system in a userspace program [21]. In FUSE, an interface named fuse_operations is provided. This interface is based on C and consists of function pointers for syscalls. What the developer needs to do is to implement a function for a syscall first, then pass the reference of the function to fuse_operations. After running the program to mount the file system, FUSE will find the necessary function pointers when a syscall is invoked through the kernel; the function that was implemented for this syscall will then be executed.

The following is a sample piece of code to illustrate the basic usage of FUSE. First, a static function named *meshFS_read()* is implemented and passed to attribute read in *meshFS_oper*, which is a static object of struct *fuse_operations*. Then, *meshFS_oper* will be passed to the function *fuse_main()* in the main function. If the user executes this program to mount the file system to a folder, whenever the operating system tries to read from this folder, the function *meshFS_read()* will be called instead of syscall *read()* in the system kernel.

```
#include <fuse.h>
static int meshFS_read(...) {
    ...
}
...
static struct fuse_operations meshFS_oper = {
    .read = meshFS_read,
    ...
};
int main(int argc, char *argv[]) {
    return fuse_main(argc, argv, &meshFS_oper, NULL);
}
```

Table 4.2 shows a list of all implemented syscalls in the MeshFS Client program. This list contains only those syscalls that are essential to the building of a functional file system. These syscalls are also sufficient for completing most operational tasks.

Table 4.2: List of Syscalls

Syscall	Purpose
meshFS_getattr	Retrieve attributes of a given node
meshFS_readdir	Read nodes from a given directory
meshFS_open	Open a file descriptor given a path or a file
meshFS_read	Read content from a file
meshFS_write	Write content into a file
meshFS_mknod	Create a new filesystem node
meshFS_unlink	Delete a file or a link from the filesystem
meshFS_mkdir	Create a new directory
meshFS_rmdir	Delete a directory from the filesystem
meshFS_chmod	Change the permissions of a node
meshFS_rename	Rename a node
meshFS_utime	Change last access and modification time of a node
meshFS_truncate	Truncate a file to a specified length

In addition to the file system interface, the MeshFS Client is also responsible for exchanging messages with MeshFS Servers, because all data or metadata need to be retrieved from the Servers. Hence, in the implementation of these syscalls, the first task that needs to be done is to set up ZeroMQ sockets to

connect to one of the MeshFS Servers. Then, the respective request message, which corresponds to the syscall, needs to be constructed and sent out through the socket. After the Server sends back a response message, the Client extracts the requested information from it, and finally sends the information back to the operating system.

4.3.3 Server

Similar to the Client, the MeshFS Server also has two responsibilities: to receive request messages from Clients, and to retrieve/update information from/to the Metaserver. The Mesh Client provides the interface to the operating system, while the Mesh Server implements most of the logic for the entire MeshFS system. Fig. 4.2 illustrates how the daemon of MeshFS Server works. At the beginning of the execution, the Server will first initialize the database connection, and register itself to the Metaserver. Then, the Server will set up and maintain a ZeroMQ server socket and listen to incoming messages from connected Clients. Once a message is received, it will be put into a message queue for processing. For each type of message mentioned above in the messaging section, there is a specific function for processing this request. In these functions, the Server either retrieves/updates data from/to the Metaserver (depending on the implementation), or forwards the request message if the data that is requested is physically stored on another Server.

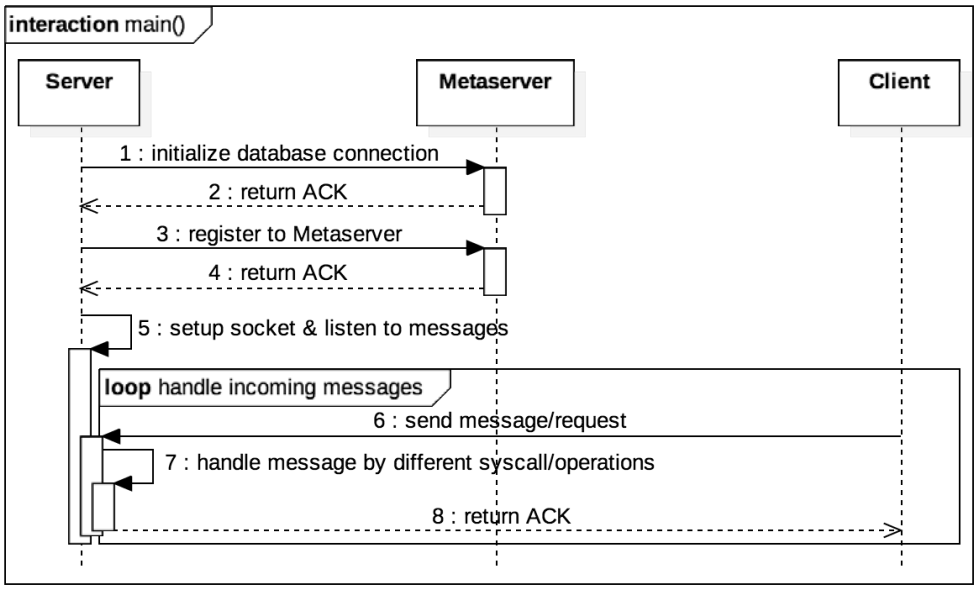

Fig. 4.2: Working Sequence of a MeshFS Server

Generally there are two types of incoming requests: syscalls and operation calls. Syscall messages can be sent from the Client or forwarded from another Server, while operation calls can be sent from the Metaserver or another Server. Table 4.3 lists the operation calls that are implemented in MeshFS. Details on how to process these requests will be discussed later, after the implementation of the Metaserver has been introduced.

Table 4.3: List of Operation Calls

Operation Call	Purpose
op_copy	Ask the server to request a copy of given file from another router
op_copy_request	Request a copy of a given file
op_checksum	Request the checksum of a given file

4.3.4 Metaserver

Different from the Mesh Client and Server, the Mesh Metaserver contains not only a C program, but also a Database for storing metadata. A mesh router that carries either one, or both, can act as a Metaserver.

Database

The Metaserver database is responsible for storing all of the meta-information for a MeshFS network, including the file metadata, directory structure, the physical location of a file and backups, and so on. The Server programs may access the database frequently; the database should therefore be able to handle concurrent or simultaneous requests. Hence, MySQL Server is adopted for the retention of metadata in this implementation. Fig. 4.3 is an entity-relationship diagram that illustrates the architecture of the Metaserver database. As the diagram shows, there are five tables in a Metaserver database:

a. The table *Server* stores information about Servers in this MeshFS network. This is also where Servers register their presence when they join the network. There are three columns in this table: a unique serverID (generated by SHA256 from its hostname and IP address), the hostname, and the IP address.

b. The table *Node* stores information about all file system nodes in MeshFS, which are directories and files. Because MeshFS has been designed for OpenWrt

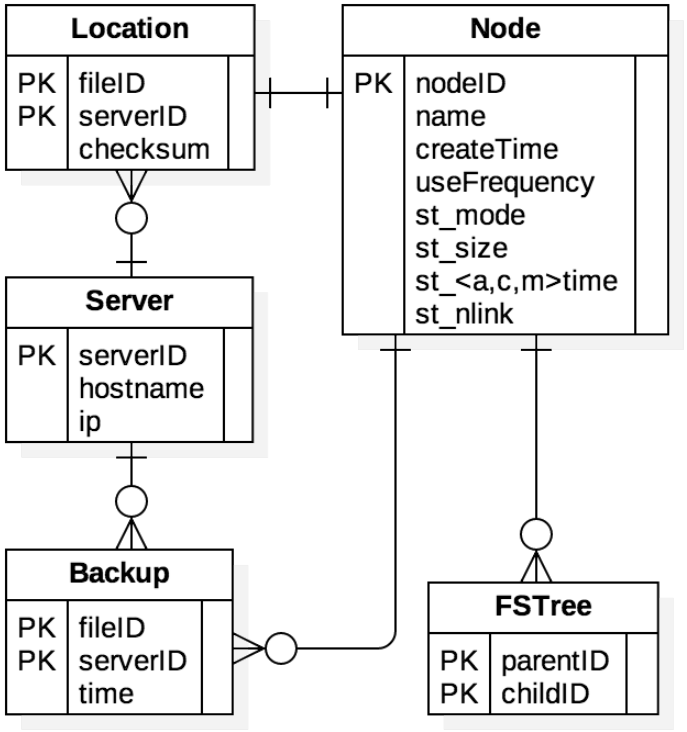

Fig. 4.3: ER Diagram of a MeshFS Metaserver Database

Linux, all of the columns that appear here are attributes from a Unix-style file system inode. Each node also has a unique nodeID, which is generated by SHA256 from its path (the path in a file system is also unique). The frequency of use is also recorded for further usages, such as for access locality analyses and predictions.

c. The table *FSTree* stores the file structure tree in MeshFS. There are only two columns: parentID and childID, both of which are referenced to nodeIDs in the table Node. Each pair that appears in this table indicates that this child is under this parent directory. Note that the node referenced by parentID must be a directory node.

d. The table *Location* stores information about a file's physical location. The table consists of three columns: fileID, serverID, and the MD5 checksum of this file. The column fileID is referenced to an entity in the table Node, and the node must be a file. The record indicates that this file is physically stored on the server that the serverID is referenced to. The checksum is also recorded so that the integrity of the file can be checked later.

e. The table *Backup* has a similar schema to the table Location: there is a pair consisting of fileID and serverID, but a timestamp appears instead of a checksum. This table stores the physical location of backups generated from a file. The timestamp is recorded for further purposes, such as for backup replacement, version control, and so on.

Storing all meta-information in one database is a centralized method, which may cause the SPOF problem. Therefore, there should normally be more than one Metaserver existing in a MeshFS network, depending on the scale. *Rsync* will be considered to be adopted to periodically synchronize the data in the Metaserver program.

Daemon

The purpose and responsibility of a Metaserver program may vary, depending on the requirements. Since a Metaserver program is an individual program that directly accesses the database and runs in isolation from MeshFS Clients and Servers, this module can easily be replaced with other programs that are implemented for different backup/recovery strategies, or even for fulfilling further different potential requirements.

In this implementation, the very basic requirement for a Metaserver is to generate file backups in case data is lost or corrupted. The maximum number of backups needed for this MeshFS network can be configured. Fig. 4.4 is the sequence diagram illustrating the strategy of a basic backup functionality, while Algorithm 2 describes the procedure.

The Metaserver program will periodically call the whole backup procedure for every file in the file system; the checking period is also configurable. The procedure can be divided into three steps. First, the original file will be checked. If the original file is lost and there is a valid backup in the network, the file will be recovered. If the file does not match the checksum in the database, which means that the file has been updated, then the checksum will be updated. Next the existing backups will be checked. If the backup does not match the checksum, they will all be re-generated together, or the oldest one will be replaced, depending on the strategy. Finally, if the number of existing backups has not met the number in the configuration, new backups will be

generated on new locations, and the respective information will be written into the Metaserver database.

Algorithm 2 Check and Generate Backup for Files

```
1:  procedure BackupCheck
2:      for each file stored in MeshFS do
3:          retrieve fileCheckSum from mesh router
4:          if fileCheckSum is changed then
5:              update fileCheckSum to database
6:          end if
7:
8:          for each backup existing in the WMN do
9:              verify correctness of backup with fileCheckSum
10:             if backup is not valid then
11:                 re-backup for file
12:             end if
13:         end for
14:
15:         while neeed more backups do
16:             generate new backup
17:             update new backup information to database
18:         end while
19:     end for
20: end procedure
```

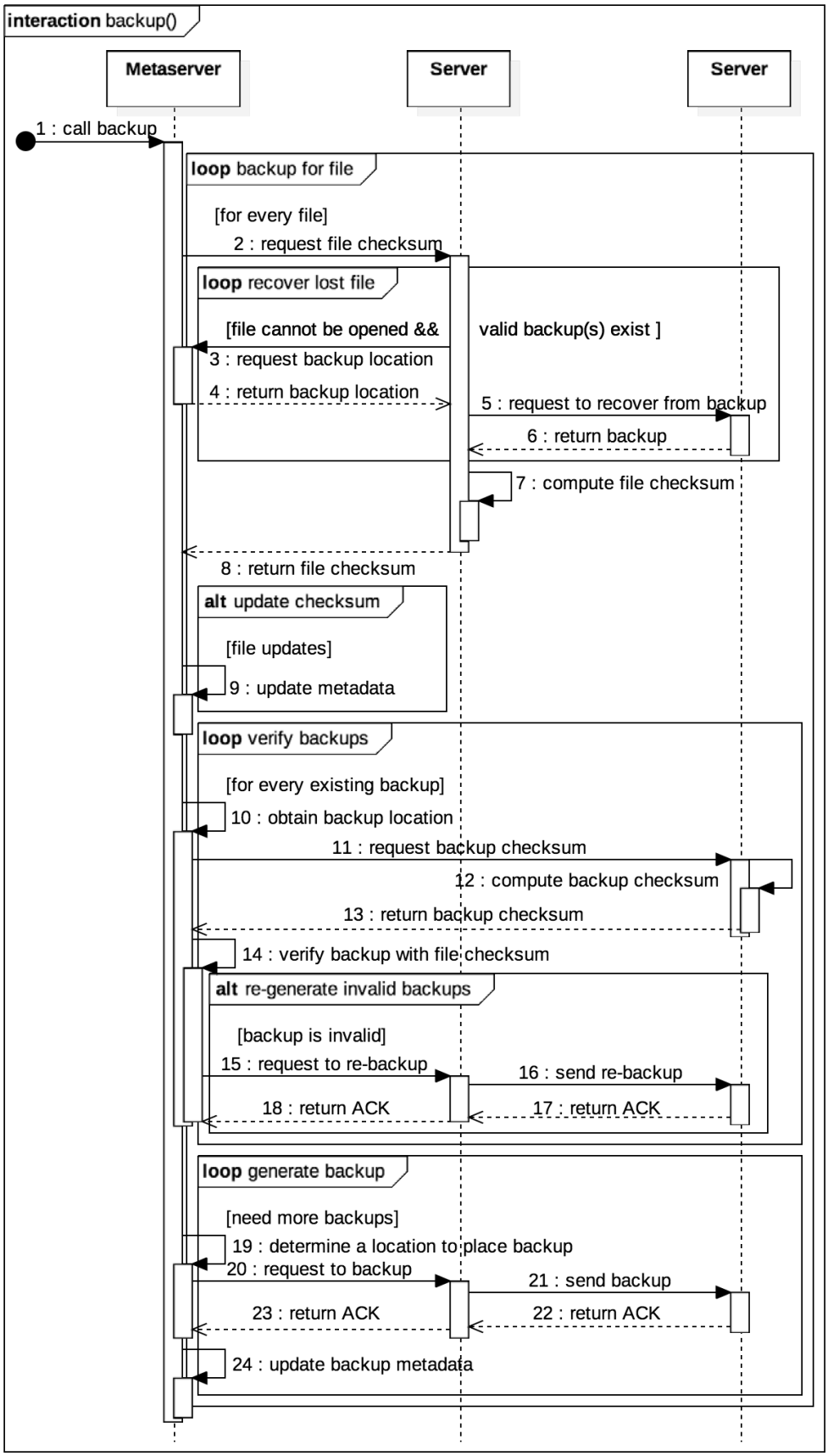

Fig. 4.4: Sequence Diagram of the Backup Function

4.3.5 Syscalls

As previously mentioned, most of the syscalls are working on a general *Request-reply* pattern: The Client sends a request to the server to which it is connected, and the server retrieves or updates the information from/to the metaserver to which it is registered. These syscalls are: *getattr()*, *readdir()*, *open()*, *mknod()*, *mkdir()*, *rmdir()*, *chmod()*, *utime()*, *truncate()*. Sequence diagram Fig. 4.5 illustrates the working sequence by which these syscalls are handled in MeshFS.

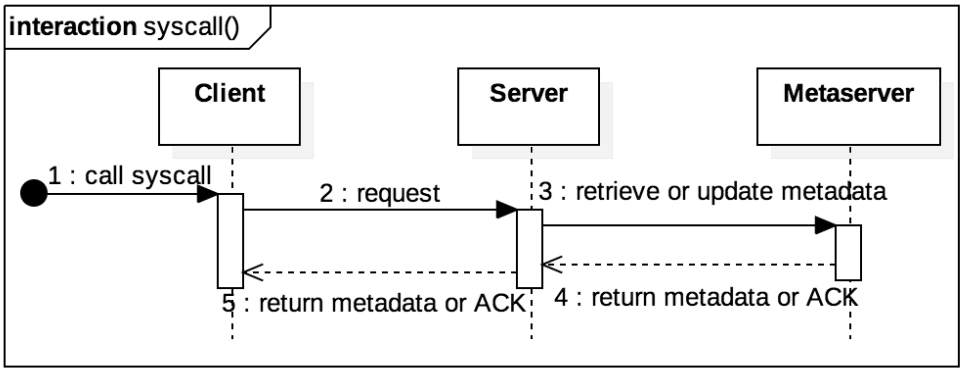

Fig. 4.5: General Working Sequence of Messaging

However, some special syscalls require much more complex processes: *read()*, *write()*, *rename()*, *unlink()*. The procedure that is performed on the MeshFS Server is described in Algorithm 3, while the entire working sequence of syscall *read()* is illustrated in the sequence diagram in Fig. 4.6.

When the operating system calls syscall *read()* to read a certain file in MeshFS, the MeshFS Client will first send a read request to the Server it is connected to. Then, the Server will locate the physical location of this file (which can be the location of the original file or the synchronized copy, depending on the strategy) from the Metaserver. If the file is physically stored on another Server, the request message will be forwarded to that router. Otherwise, the Server will check this file and read the content. If the file is lost or corrupted, the recovery procedure will be triggered. Then, the required content will be sent back to the Client, and metadata such as last access time and frequency of use will be updated.

The operation sequence of syscall *write()* is similar to that of *read()* (Algorithm 4 and Fig. 4.7): when the operating system calls it, the Client will send the

Algorithm 3 *Read()* Procedure on Mesh Server

Input: $fileID$, $path$, $offset$, $size$
Output: $fileContent$

```
1: procedure MeshFS_Read
2:     fileLocation ← retrieveFileLocation(fileID)
3:
4:     if fileLocation is this router then
5:         if file cannot be opened then
6:             try to recover file from backup
7:         end if
8:
9:         if file is valid then
10:            read file content into fileContent
11:        else
12:            write error message into fileContent
13:        end if
14:    else
15:        forward read request to the router where file is located
16:    end if
17:
18:    update Metadata database
19: end procedure
```

Algorithm 4 *Write()* Procedure on Mesh Server

Input: $fileID$, $path$, $offset$, $size$
Output: $status$

```
1: procedure MeshFS_Write
2:     fileLocation ← retrieveFileLocation(fileID)
3:
4:     if fileLocation is this router then
5:         setup socket to receive fileContent
6:         write fileContent into file
7:     else
8:         forward write request to the router where file is located
9:     end if
10:
11:    update Metadata database
12: end procedure
```

request message, and the Server will enquire about the location of the file from the Metaserver. If the file is physically stored on another mesh router, the message will be forwarded. Otherwise the content will be written into the file, and the corresponding metadata (e.g., size, last access time, last modification time, etc.) will be updated. Then an acknowledgement will be returned to the

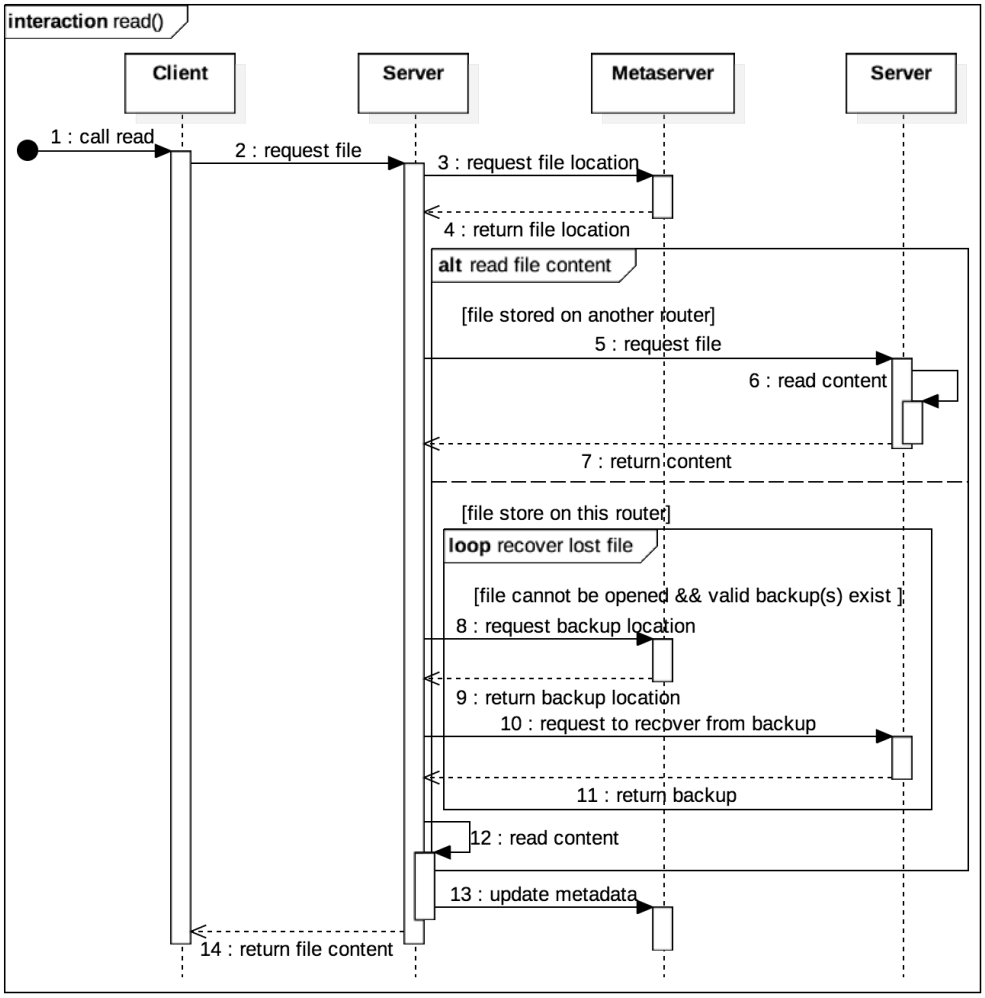

Fig. 4.6: Sequence Diagram of syscall *read()*

Client. The difference between *write()* and *read()* is that there is no need to perform an integrity check and data recovery in *write()*.

Since syscall *rename()* can be used to rename both a directory and a file in a file system, the implemented *rename()* also needs to handle the use case accordingly (Algorithm 5 and Fig. 4.8). After the Server receives a rename request from a Client, it will first update the node name in the Metaserver database. Then, the Server will check the node type. If the node is a file, the remaining procedures will be similar to those of syscall *write()*: locate the file and rename it on the physical storage, or forward the message if it is stored on another router, then location and backup information will be updated in the Metaserver database. If the node is a directory, the Server needs to recursively update the metadata for all of the descendants of this node, because all of the

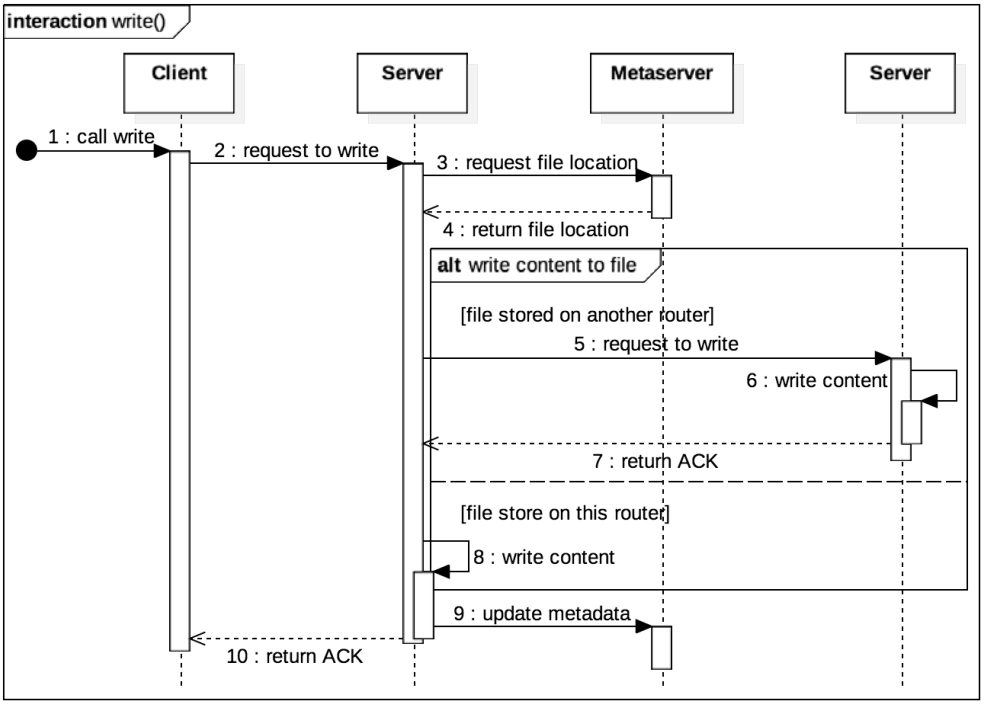

Fig. 4.7: Sequence Diagram of syscall *write()*

Algorithm 5 *Rename()* Procedure on Mesh Server

Input: *path*, *newPath*
Output: *status*

```
1:  procedure MeshFS_Rename
2:      nodeID ← getNodeID(path)
3:      newNodeID ← getNodeID(newPath)
4:      update nodeID to newNodeID in database
5:
6:      if node is directory then
7:          recursively rename all descendants
8:      else if node is file then
9:          fileLocation ← retrieveFileLocation(nodeID)
10:
11:         if fileLocation is this router then
12:             rename file on this router
13:         else
14:             forward rename request to the router where file is located
15:         end if
16:     end if
17: end procedure
```

paths of its descendants will have been modified after the renaming, and a nodeID will have been generated from the absolute path in MeshFS.

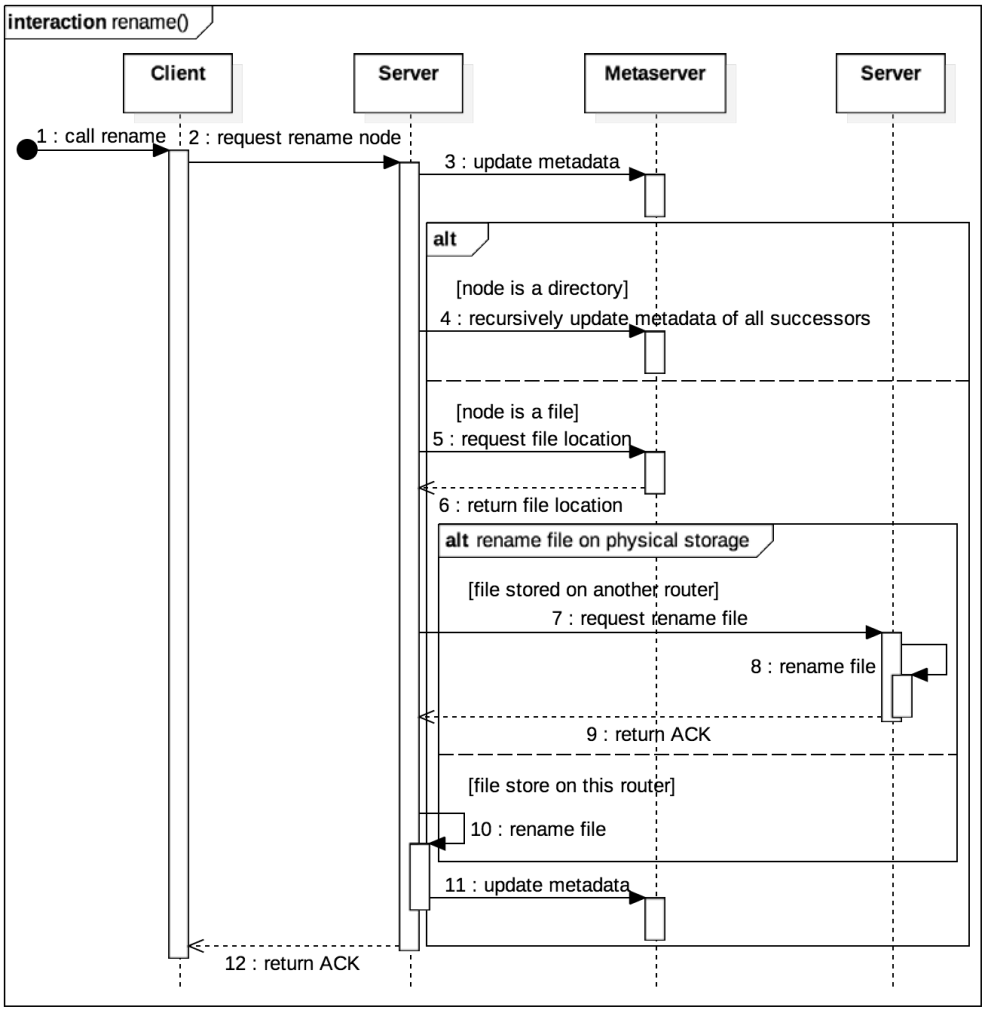

Fig. 4.8: Sequence Diagram of syscall *rename()*

Algorithm 6 *Unlink()* Procedure on Mesh Server

Input: *path*
Output: *status*

```
1: procedure MeshFS_Unlink
2:     nodeID ← getNodeID(path)
3:     delete all entries with nodeID from database
4:
5:     fileLocation ← retrieveFileLocation(nodeID)
6:     if fileLocation is this router then
7:         delete file from this router
8:     else
9:         forward unlink request to the router where file is located
10:    end if
11: end procedure
```

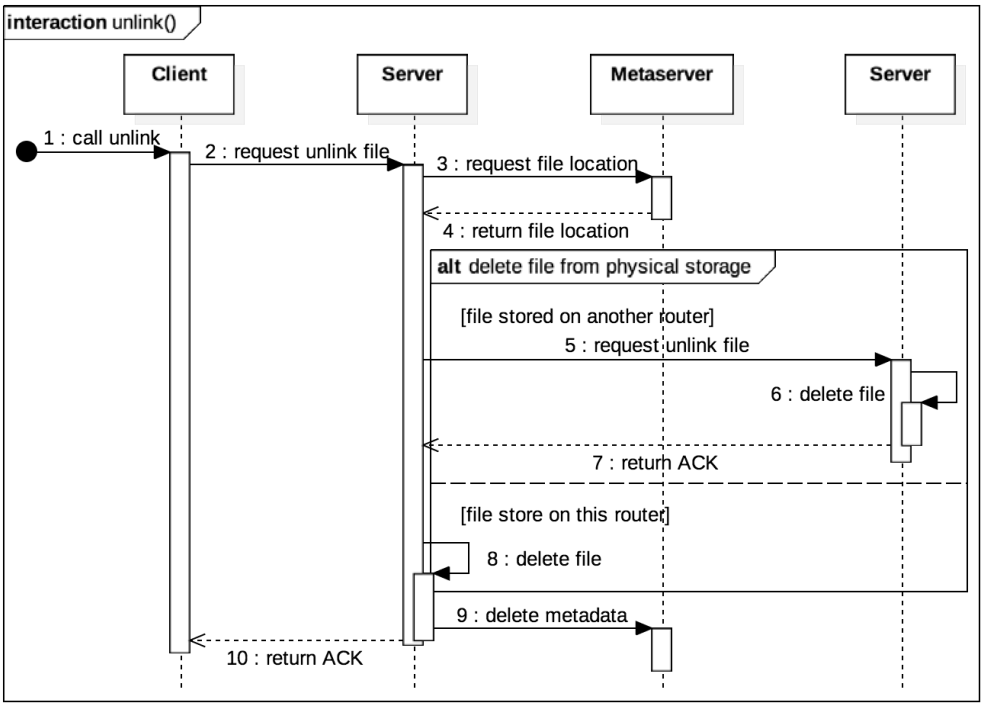

Fig. 4.9: Sequence Diagram of syscall *unlink()*

Because there is another syscall *rmdir()* for deleting directories, syscall *unlink()* only deletes files in a file system. Hence, its implementation is almost the same as that of syscall *write()* (Algorithm 6 and Fig. 4.9). The only difference is that *unlink()* deletes files and metadata from physical storage and the Metaserver database instead of writing new content into them.

4.4 Demonstration

In order to demonstrate the features and functionalities of MeshFS, a website was built with multimedia resources in MeshFS to evaluate its performance. The website consists of an index file in HTML, a 720p video file, and other supplementary files such as images, JavaScript sources, and others. The special feature of accommodating websites on MeshFS is that although all files and resources are physically stored on different routers, MeshFS makes them appear to be under the same directory as that of web server applications running on upper layers (e.g., uHTTPd, Lighttpd, Apache, Nginx).

Fig. 4.10: Demo Website with Multimedia Resources Hosting on MeshFS

In the demonstration, a 720p video file with a size of approximately 12 MBytes (12623948 Bytes) and a length of 51 seconds is used to determine whether this file system can satisfy commercial requirements, such as seamlessly playing

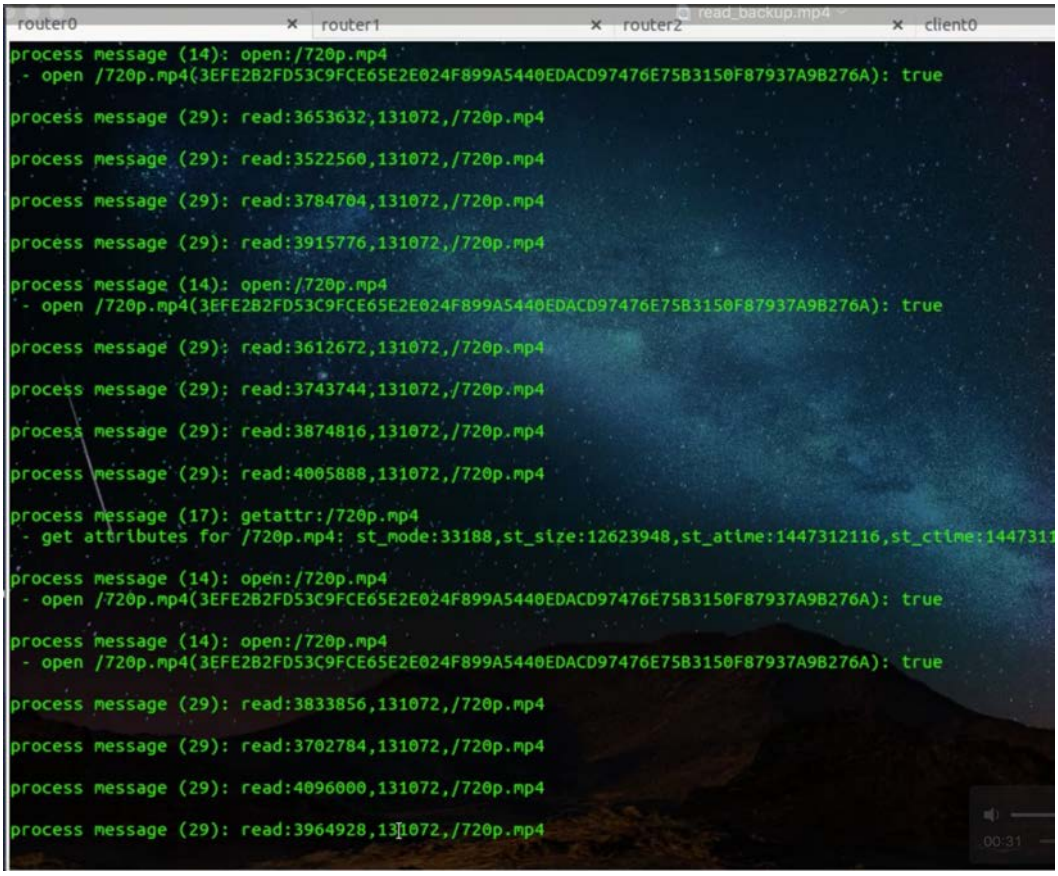

Fig. 4.11: MeshFS Reading Data when Video is Playing

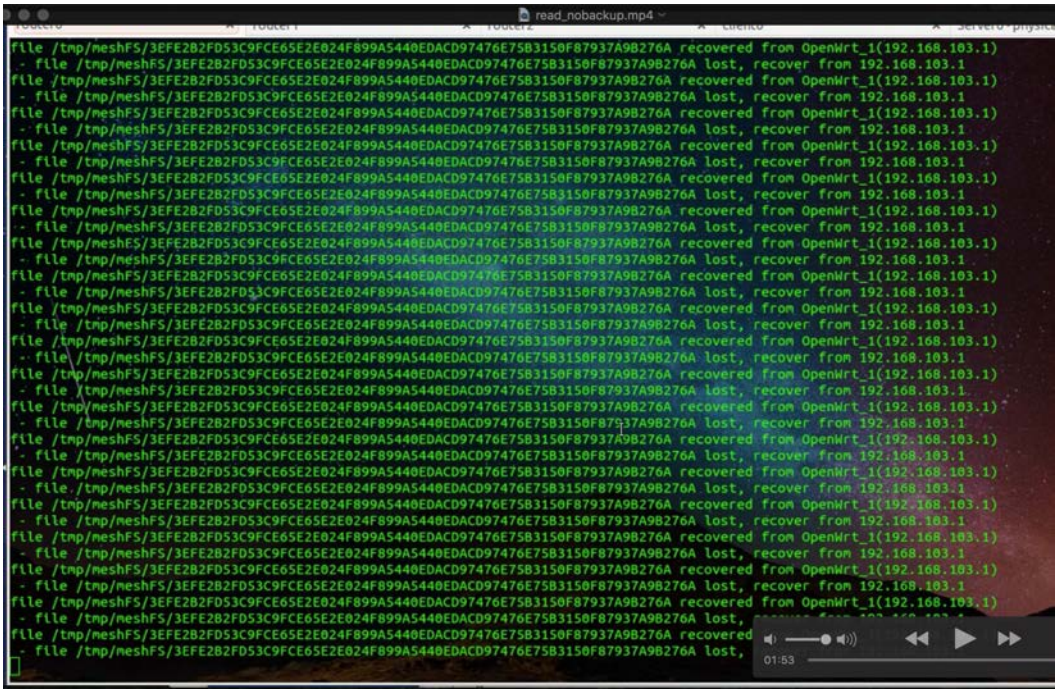

Fig. 4.12: Data Lost without Enabling Backup Function

Fig. 4.13: Video Stopped at the Corrupting Point without Backup

Fig. 4.14: Video Continued at the Corrupting Point with Backup

video advertisements in shopping malls without disturbances or failures. This video file is physically stored on one router several hops away from the router that acts as the web server. The exact number of hops depends on the network

topology and routing tables generated by the WMN itself. At first, it starts playing the video through the website without enabling the backup & recovery function and trying to corrupt the file in the middle of the video. As shown in the upper part of Fig. 4.16, the video can be played very smoothly at first. However, after all buffered data is played out, the film will be forced to stop playing at the corrupted point as expected because the file has been corrupted, and no more data will be buffered.

Fig. 4.15: Corrupted Data Recovered by MeshFS with Backup/Recovery Enabled

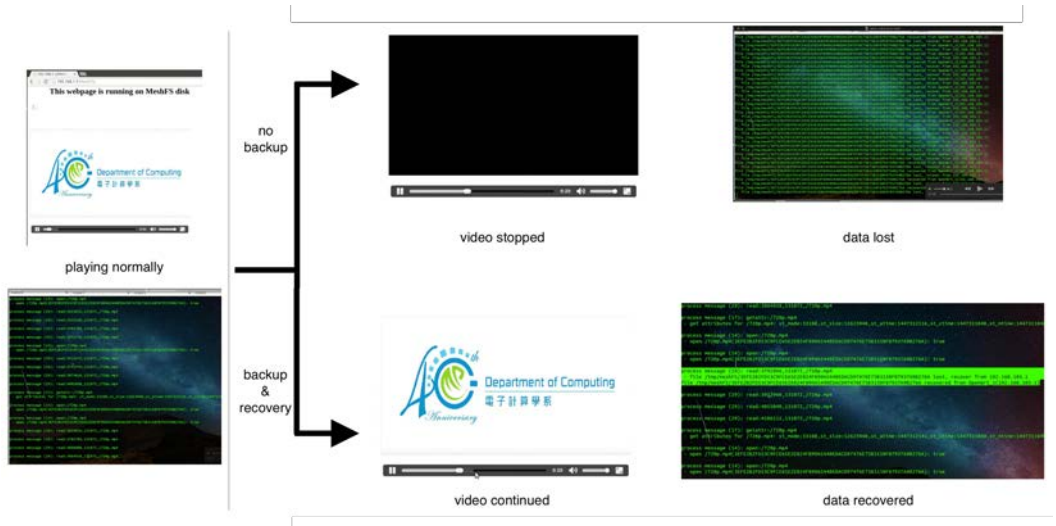

Fig. 4.16: Summary of MeshFS Demo

Next, the backup & recovery function is enabled to determine whether it can conquer the corruption of the file without affecting the playing of the video. In this case, MeshFS will automatically replicate the file on another router as a redundancy, keep the checksum recorded, and periodically check the consistency of the file. Then, the video file is played again, and we tried to

corrupt the file at the same point as before. This time, the streaming buffer will first stop at the corrupted point due to the lost data; however, only a few seconds later the streaming buffer will again start to grow, and this slight pause will have no influence at all on the playing process (the lower part of Fig. 4.16). The reason behind this is that MeshFS detected the corrupted data when it tried to read the file, and then instantly performed data recovery from backups. It can be observed that MeshFS was able to quickly detect and recover data without any disturbances or failures to affect the performance of the video playing in 720p resolution. Fig. 4.16 summarizes the entire flow of playing the video from MeshFS with/without enabling the backup/recovery function.

4.5 Evaluation & Limitation

As was introduced at the begining, the main objective of MeshFS is to utilize scattered computing and storage resources on an existing wireless mesh network. The most important requirements for MeshFS are simplicity and stability. Hence the access speed of MeshFS might not be as fast as that of other existing DFSs. MeshFS only focuses running on mesh routers, which provides very limited computing capacity. Other popular distributed computing models, such as MapReduce, might provide overwhelming to mesh routers.

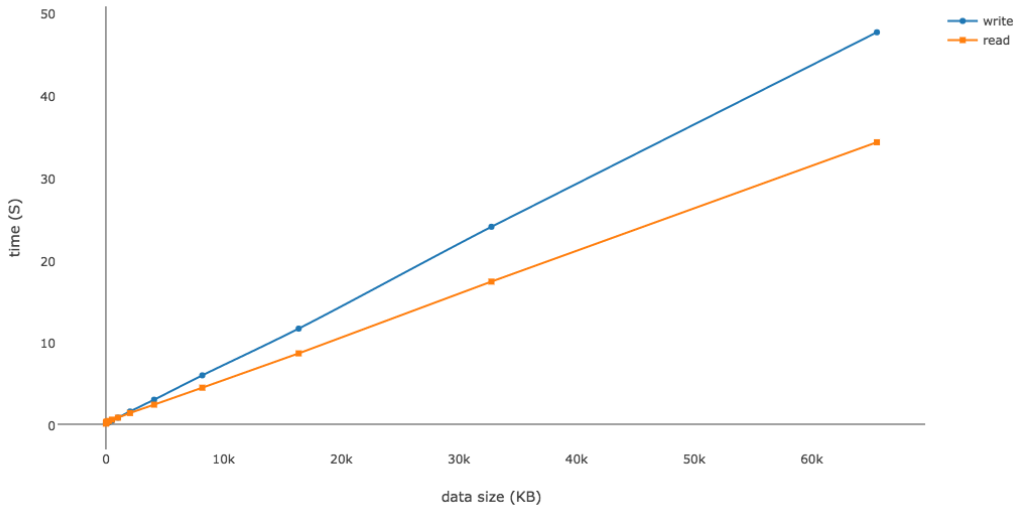

Fig. 4.17: Relationship between Data Size and Time in MeshFS

As shown in Fig. 4.17 and Fig. 4.18, the I/O (*read*() & *write*()) performance of MeshFS is illustrated. It can be observed that data transfer times are

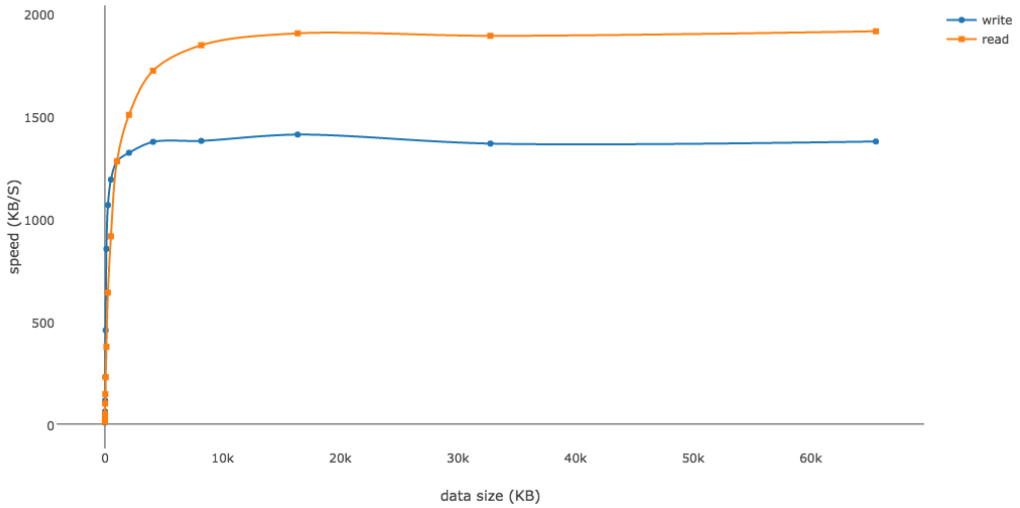

Fig. 4.18: Relationship between Data Size and Transfer Speed in MeshFS

growing linearly along with an increase in the size of the data for both *read*() and *write*() in MeshFS. From the relationship between data size, transfer time, and transfer speed ($speed = datasize/time$), it can be predicted that transfer speed should be constant due to the linear growth over time. In Fig. 4.18 it is confirmed that the transfer speed becomes stable after the data reaches 2048 KB in size. The average transfer rate of *read*() is approximately 1.9 MB/S, while the average transfer rate of *write*() is approximately 1.4 MB/S. As discussed earlier, although these two numbers indicate that MeshFS cannot provide high speed I/O as a DFS, the main idea of this system is to gather and utilize scattered resources among mesh routers to provide an integrated storage interface with fault-tolerance functionality.

The backup/recovery functionality of MeshFS was demonstrated in the previous section. To evaluate the performance of this functionality, the data throughput during the playing of the video will be illustrated to analyze the difference between enabling and disabling this function, and to determine whether the requirement for continuous playing can be met.

As shown in Fig. 4.19, the data flows recorded during the playing of the video appear to differ between the enabling and disabling of the backup/recovery function. The video will be corrupted at the point of the 20th second in this experiment. During the first 20 seconds, both tests behave similarly: the data transfer speeds are at the middle level during playing. At the point of the 20th second, the speeds drop rapidly due to the loss of data. If the backup function

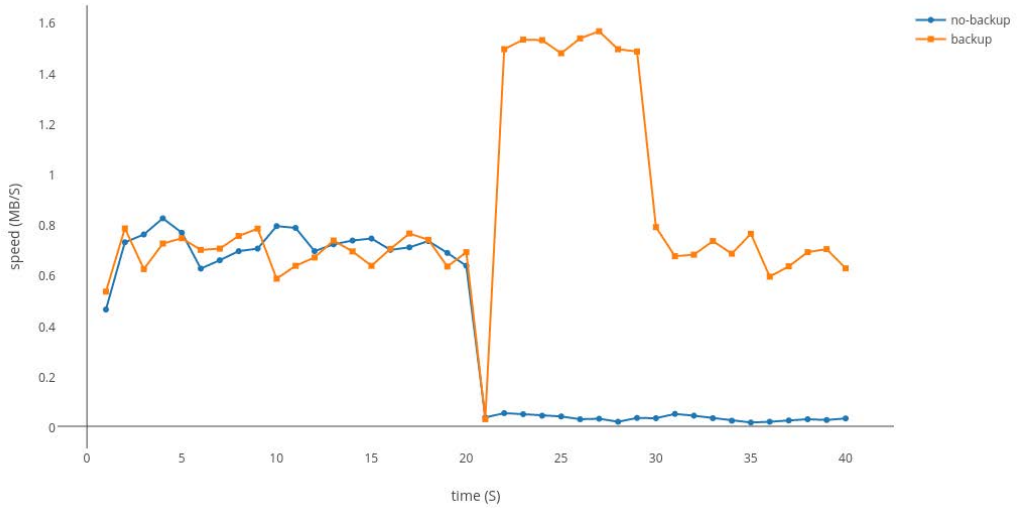

Fig. 4.19: Comparison of the Flow of Data when Playing Videos in MeshFS with the Backup Enabled/Disabled

is disabled, the transfer rate will remain very low because only a few messages are exchanged afterwards. However, if the backup function is enabled, the transfer rate will suddenly rise, then drop to a normal level after a few seconds. The reason for this is that after MeshFS has detected the loss of data at the point of the 20th second, it will start to invoke the data recovery process by transferring previous backups from other routers. After a successful recovery, the transfer rate will drop back to the middle level and the video will keep playing without interruption.

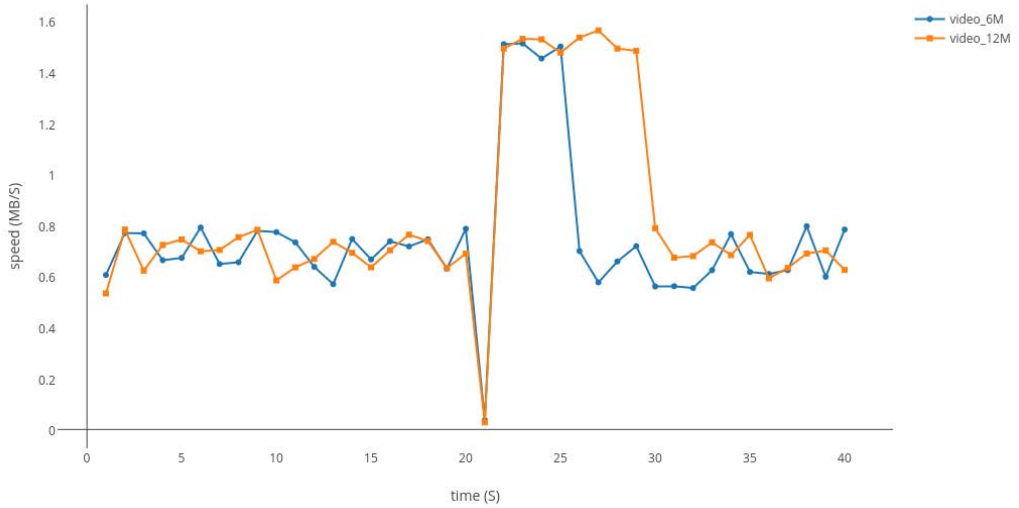

Fig. 4.20: Comparison of the Flow of Data when Playing Videos of Different Sizes in MeshFS with the Backup Enabled

Fig. 4.20 gives a comparison of data transfer speeds when playing videos of different sizes with the backup/recovery function enabled. One size is the aforementioned 12 MB 720p video and another is a half-length copy of it, the size of which is also approximately half that of the original. Both of the videos will be corrupted at the point of the 20th second. It can be observed that the data flows of both videos appear to be similar: both behave in a relatively stable manner during the first 20 seconds, then drop down to nearly zero when reaching the point of corruption. After that, both rise rapidly as the recovery process starts, then drop down again into a normal rate. The widths of the peaks on the graph depend on the size of the videos. Hence, about twice as much time is needed to recover the flow of data for a 12 MB video than for a 6 MB video. Fig. 4.21 and 4.22 illustrate the relationship between recovery time and speed to the size of video files. It can be observed from the graph that recovery time grows linearly with the increase in file size, while the speed of recovery remains very stable for all sizes of tested video files.

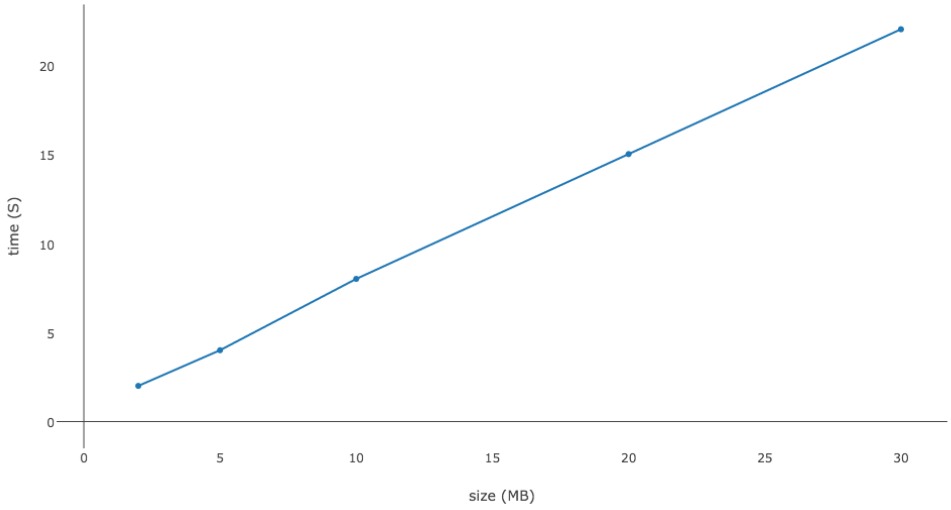

Fig. 4.21: Relationship between Recovery Time and Size of Video Files

MeshFS provides a file system interface and storage spaces for the operating system and upper-layer applications; however, how to utilize the file system is fully dependent on the operating system. This limitation arises at calling syscall *read()* and *write()*. The operating system does not read/write all content in one syscall. Instead, the operating system calls *read()*/*write()* for a chunk once a time. When the operating system invokes these two syscalls, the system kernel will pass on the size of the content to the syscall. For example, if there is a file with the size of 1024KB stored in MeshFS, and the size of the chunk for the operating system is 16KB, then the kernel will call syscall *read()*

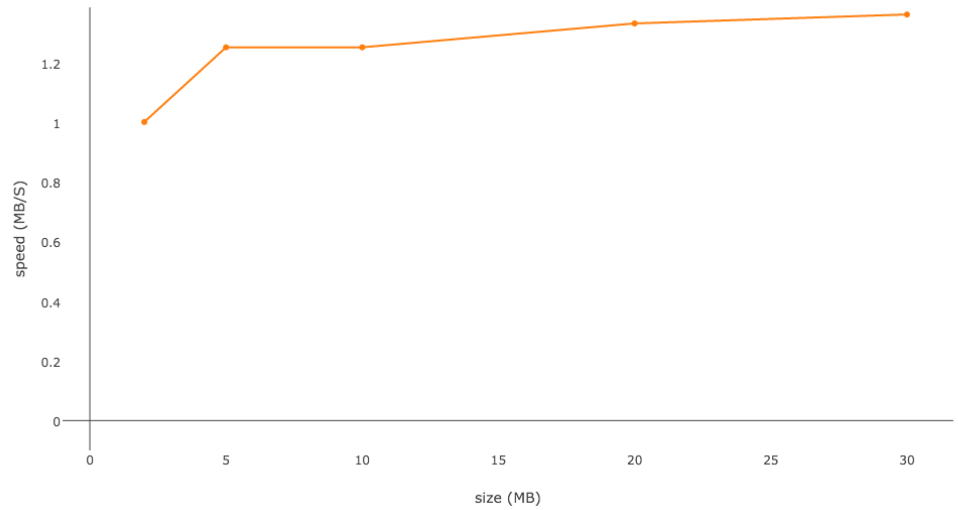

Fig. 4.22: Relationship between Recovery Speed and Size of Video Files

64 times to read the entire file. The smaller the chunk size, the more messages there will be. Since one message is generated and sent out every time the syscall is called, extra calls will cause extra network traffic, more SQL queries, and more time for the Server to process the message.

In OpenWrt Linux, the maximum size of the chunk for syscall *read()* is 128KB (131072 Bytes), which means that one message is required to read every 128KB of data. Moreover, the size of the chunk for a syscall *write()* varies depending on the caller program. Testing results show that the system tool *cp*, which is a preloaded command for copying files in the shell, only uses a 4KB chunk size to call a syscall *write()*, which is very slow compared to a *read()*. This limitation can partially be overcome by replacing *cp* with another system tool *dd* for I/O operations, and mounting MeshFS with the option "-o big_writes". This operation allows FUSE to call a *write()* at a 128KB chunk size, which is equivalent to a *read()*. 128KB is the maximum chunk size that current versions of OpenWrt and FUSE can support. Modifications of the source code of the system kernel and FUSE are required in order to overcome this barrier.

Table 4.4 shows the difference between using the normal mode or using the "big_writes" mode for a syscall *write()*. To transfer the aforementioned 720p video file with a size of 12 MBytes (12623948 Bytes), the normal mode takes 265.16 seconds while the "big_writes" mode takes only 9.97 seconds. The "big_writes" mode is 26.6 times faster than the normal mode, while the size of the message unit is 1/32 that of the normal writing mode. To transfer

Table 4.4: Comparison between the Normal Mode and the "big_writes" Mode for *write()*

Size	Message Unit Size	Time	Speed
12623948 Bytes (12 MBytes)	4096 Bytes (4 KB)	265.16 S	46.49 KB/S
	131072 Bytes (128 KB)	9.97 S	1236.52 KB/S
35027658 Bytes (33.4 MBytes)	4096 Bytes (4 KB)	677.03 S	50.52 KB/S
	131072 Bytes (128 KB)	23.57 S	1451.28 KB/S

another video file with a size of 33.4 MBytes (35027658 Bytes), the normal mode takes 677.03 seconds, while the "big_writes" mode takes only 23.57 seconds to transfer this file. The "big_writes" mode is 28.7 times faster than normal mode. The data transfer speed of *write*() matches the recorded value in Fig. 4.18.

Another limitation of this implementation is that only a basic strategy for the backup/recovery function has been implemented to prove that this system can work on mesh routers. As previously mentioned, this strategy module is actually a daemon program that is run on Metaservers, and can be easily replaced with more advanced programs.

4.6 Conclusion

This chapter has presented MeshFS, a distributed file system for cloud-based wireless mesh networks with automatic backup/recovery as the fault-tolerance, and a Unix/Linux mountable interface for accessing the file system. With respect to speed, although MeshFS is not as competitive with popular DFS solutions, it focuses on integrating mesh routers together to better utilize scattered storage resources from mesh routers in a wireless mesh network. It is simple, lightweight, cooperative, and has essential backup/redundancy functionalities.

In the next chapter of this research study, it is planned to develop a group-based adaptive data management mechanism for cloud-based wireless mesh networks, which is able to replicate data that can then be shared by multiple mesh clients to achieve an optimal access cost, instead of simply backing up data. These replications are consistent and entirely synchronized, thus allowing users to access this data from their closest location to reduce access

overhead. The mechanism should be able to adaptively decide how many replications are needed and where these replications should be placed in order to minimize the total cost of accessing and transferring the data. MeshFS will be adopted as the foundation for providing storage services and supporting data replication/moving operations.

5 Group-based Adaptive Data Management Mechanism

The third part of this project is to investigate a group-based adaptive data management mechanism. In this scenario, a file may be shared/owned by multiple mesh clients. Hence the file cannot simply be moved along with a single mesh client. Much like with the person-based adaptive data management mechanism, the purpose of group-based adaptive data management mechanism is to enhance the speed with which mesh clients can access stored files, and decrease the cost/overhead involved in transmitting data. With the aim of satisfying these objectives, the mechanism tries to find an optimal solution to place the files in the mesh routers. A typical scenario with such requirements is called the e-book scenario.

5.1 E-book Scenario

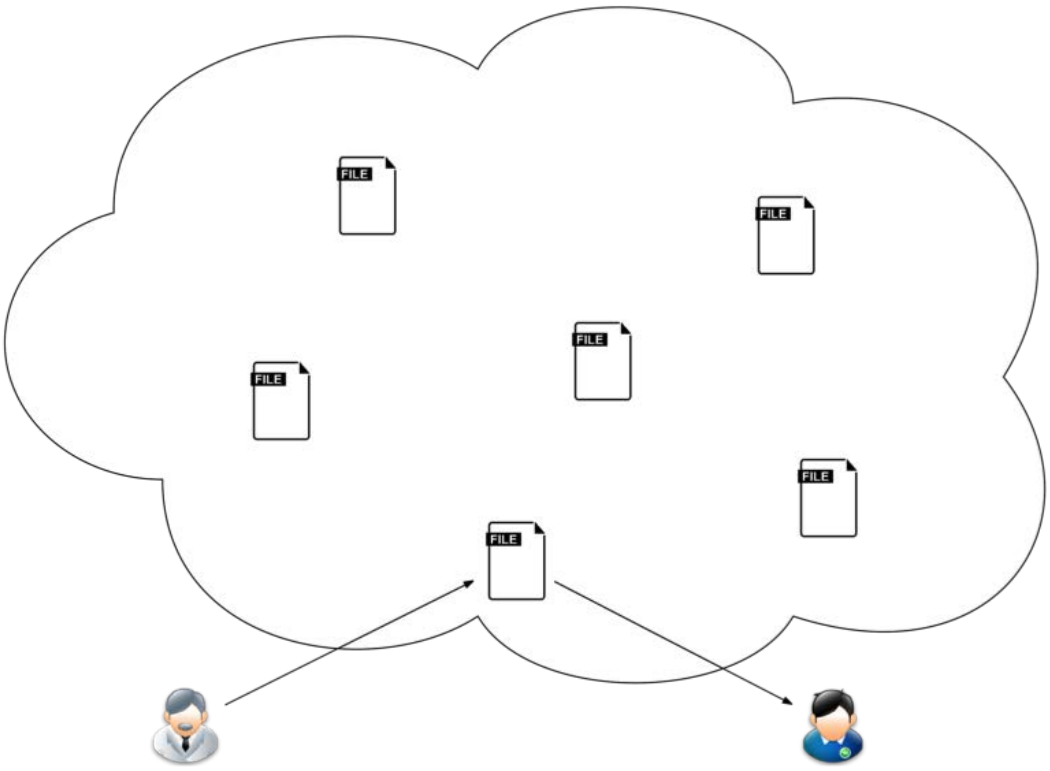

Fig. 5.1: E-book Scenario

The e-book scenario normally occurs in educational environments: in colleges or schools, teachers often need to upload certain digital resources (e.g., electronic textbooks, notes, assignment papers, etc.) to cloud storage so that students can access/download these resources to their own devices for viewing or printing (Fig. 5.1).

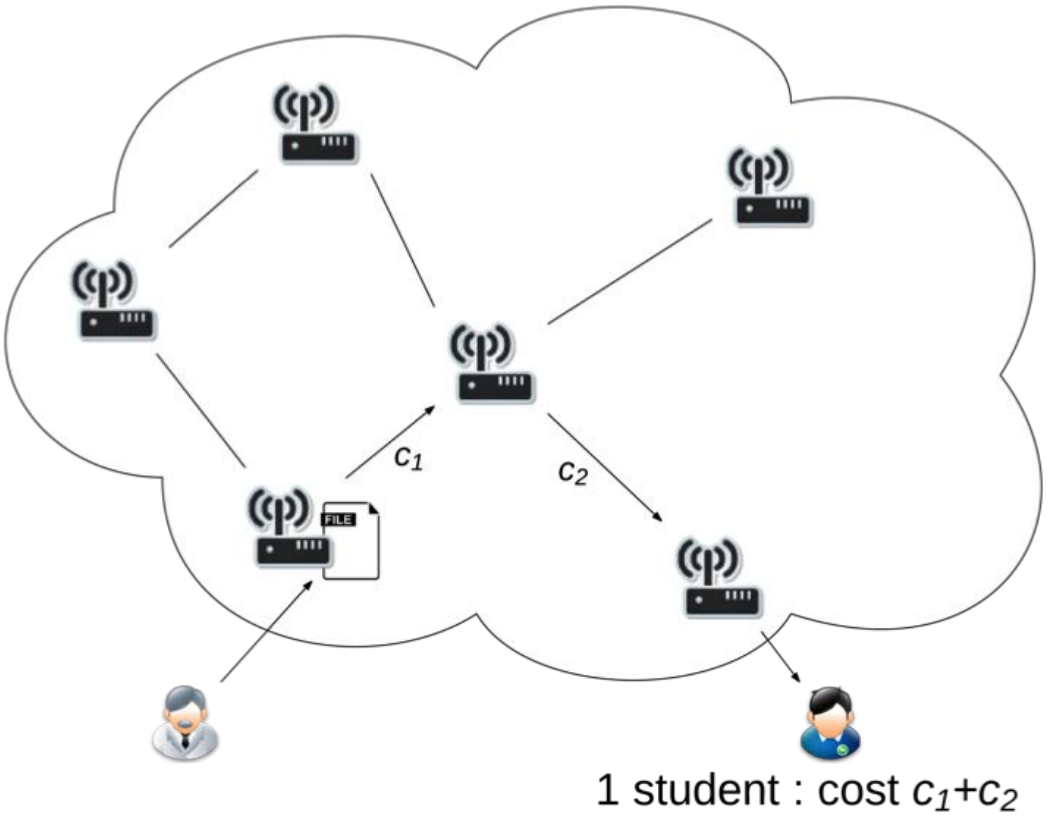

Fig. 5.2: E-book Scenario: Cost for 1 Student

Physically, the cloud storage involved in this scenario consists of mesh routers with storage capacity from the wireless mesh network. After uploading, an electronic resource (file) is physically stored on a certain router, and its storage information is registered in the database so the system can become aware of its location. When a user wants to download this file, the file will be transferred through the routing path on the mesh network at a downloading cost (e.g., $c_1 + c_2$) (Fig. 5.2).

If multiple users share the same interest in a file (e.g., 100 students from the same class want to download the lecture notes that were just uploaded), the downloading process is repeated, resulting in a high downloading cost (e.g., totalling $100(c_1 + c_2)$ for all of the students in the example) (Fig. 5.3).

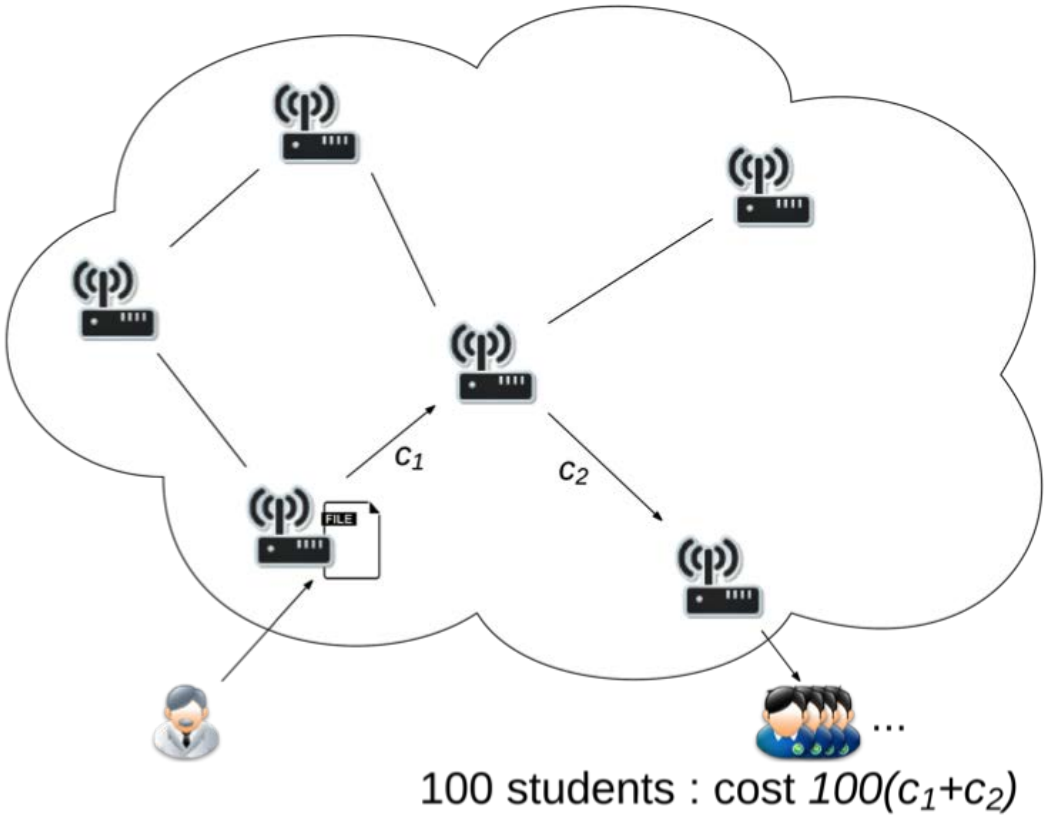

Fig. 5.3: E-book Scenario: Cost for 100 Students

Although the students are not necessarily connected to the same router in the network, they generally tend to remain close. The phenomenon of frequent accessing of the same data is called *temporal locality*. It is likely that students will share a part of the routing path, and repeated transfers over this path are redundant and wasteful. If the file can be moved/cached at routers near the students, network resources can be utilized more efficiently. In Fig. 5.4, the file was moved to the next mesh router, which is one hop closer to the group of students. This one-time movement only costs c_1. Students download this file from the new location, which costs $100c_2$. Therefore, the total cost is $c_1 + 100c_2$, which saves $99c_1$ for 100 students to download this file.

As shown in Fig. 5.5, students may spread across campus to access the materials provided by the lecturer. Hence, access or downloading requests for the file may come from all over the WMN. The ultimate goal of the group-based adaptive data management mechanism is to locate one or more of the most suitable mesh routers, based on predicted access requests, to store the files so that the global cost of downloading can be minimized.

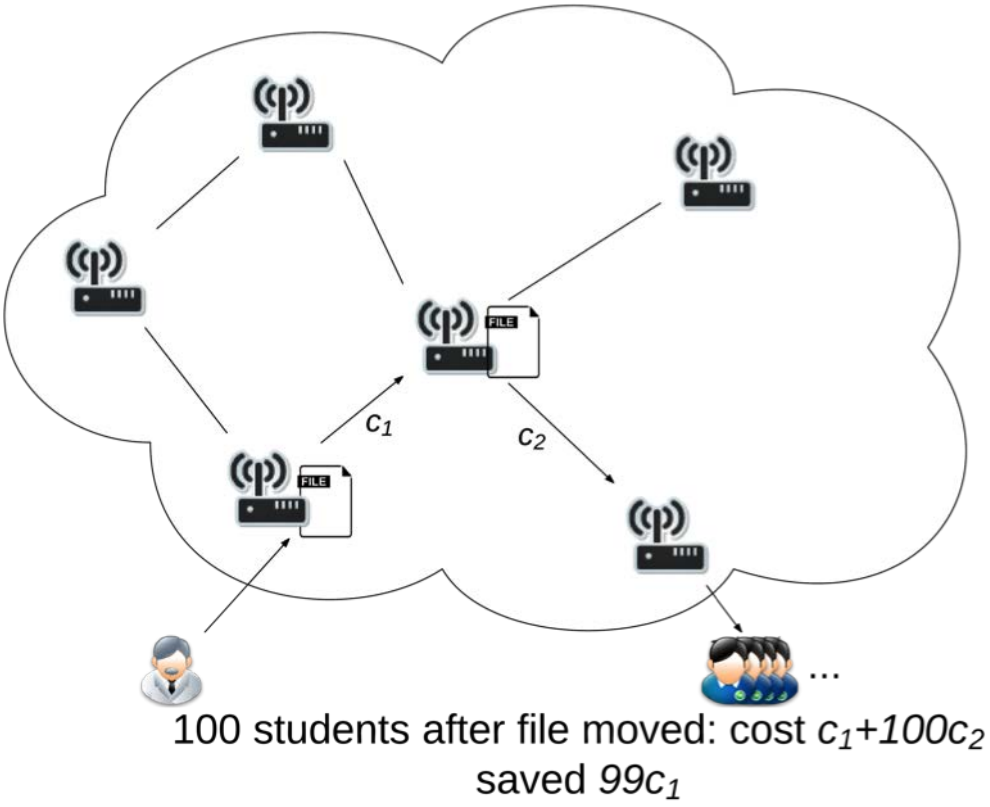

Fig. 5.4: E-book Scenario: Saving $99c_1$ Cost after the File is Moved

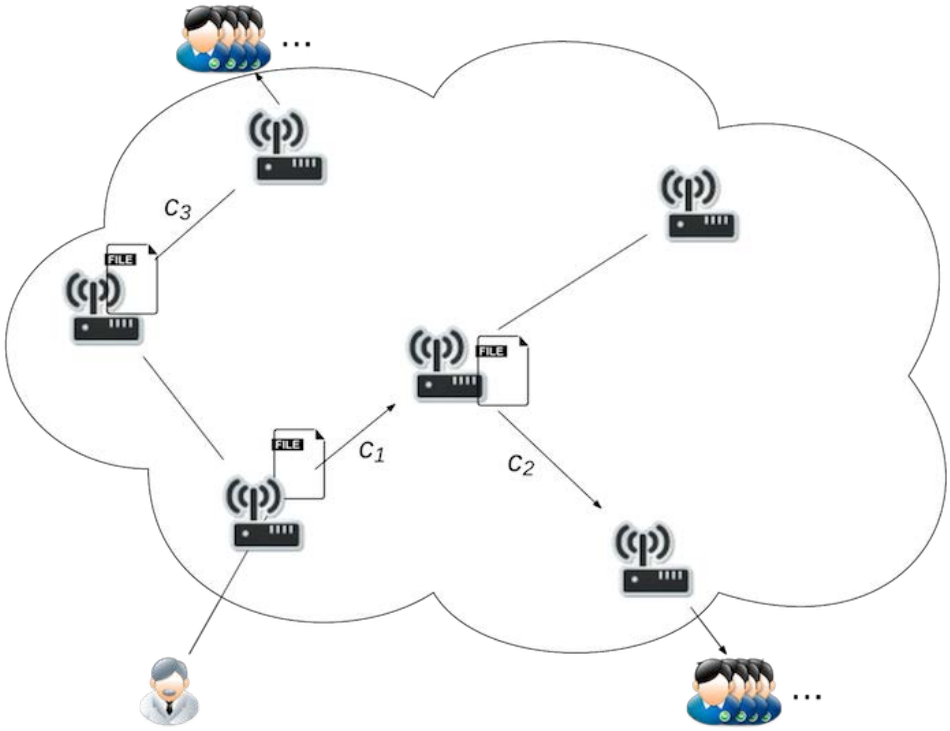

Fig. 5.5: E-book Scenario: Multiple Copies of the File are Spread in the WMN

The algorithm and mechanism to be introduced in the next sections will be implemented and integrated into the MeshFS distributed file system proposed in Chapter 4 as a storage strategy. Hence, all data or file replicas will be stored on mesh routers, and the decision of choosing physical locations will be handled internally by the file system, which is totally transparent to upper layer applications such as the web application for teachers/students to upload/download files. By storing data on mesh routers, external servers or data centres can be eliminated, thus the cost of purchasing, installing, and maintaining of such expensive equipment can be saved.

5.2 Model & Algorithm

5.2.1 Model

A Wireless Mesh Network deploying MeshFS consists of n servers and m files: the n servers, which are routers providing storage space to allow clients to access files, are denoted as $\mathbf{R} = [r_1, r_2, \dots r_n]$, and each server is equipped with memory space $\mathbf{M} = [m_1, m_2, \dots m_n]$; the m files stored on servers are denoted as $\mathbf{F} = [f_1, f_2, \dots f_m]$, and each file has a size of: $\mathbf{S} = [s_1, s_2, \dots s_m]$.

For each file f_i, there are extra metadata stored in MeshFS: 1. predicted access based on past access records: $(nodeID, serverID, timestamp)$; 2. the expected access vector in a period predicted from past records: $\mathbf{A} = [a_1, a_2, \dots a_n]$, where a_i indicates the number of requests generated by the router i (e.g., if access requests are coming from router 2, router 3, router 1, then router 3 again, then $\mathbf{A} = [1, 1, 2]$); 3. the file has k current copies at different locations (servers): $\mathbf{L} = [l_1, l_2, \dots l_k]$, where l_i indicates whether the router stores the file (e.g., if the file has two copies on router 2 and router 3, then $\mathbf{L} = [0, 1, 1]$).

For the entire wireless mesh network, there is a cost matrix:

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$

where c_{ij} is the cost that indicates the time needed to transfer file f, and can be determined (e.g., by $s/bandwidth$). Other parameters may also be introduced (e.g., loss rate, delay, etc.) to enhance the accuracy of the cost.

5.2.2 Problem Definition

The problem can be formulated as:

> Given $\mathbf{A} = [a_1, a_2, \dots a_n]$, n, $\mathbf{L} = [l_1, l_2, \dots l_k]$, k, and C, find a new ($\mathbf{L}' = [l'_1, l'_2, \dots l'_{k'}], k'$) so the total expected cost can be optimized.

The total expected cost consists of two parts:

1. Expected cost for every server to download the nearest copy from $(\mathbf{L}', k')$

$$\sum_{i=1}^{n} a_i \cdot c_{il'_{nearest}}$$

2. Cost to transfer the current copies or make new copies from $(\mathbf{L}, k)$ to $(\mathbf{L}', k')$

$$\sum_{i=1}^{k'} c_{l'_i l'_{nearest}}$$

5.2.3 Proposed Solution & Algorithm

Proof of NP-Completeness

First, transform this optimization problem into a decision problem:

> Given $\mathbf{A} = [a_1, a_2, \dots a_n]$, n, $\mathbf{L} = [l_1, l_2, \dots l_k]$, k, and $\mathbf{C}$, is there a new ($\mathbf{L}' = [l'_1, l'_2, \dots l'_{k'}], k'$) such that makes its total expected cost less than before?

If there is a certificate $(\mathbf{L}' = [l'_1, l'_2, \dots l'_{k'}], k')$, it can be verified by computing its total expected cost:

$$\sum_{i=1}^{n} a_i \cdot c_{il'_{nearest}} + \sum_{i=1}^{k'} c_{l'_i l'_{nearest}}$$

which is in polynomial time, so the decision problem is **NP**.

The decision problem can be reduced to the **subset sum problem**, which is **NP-hard**.

So the decision problem, as well as the optimization problem, is **NP-Complete**.

Brute-force Solution

Intuitively, this problem can be directly solved by simply traversing through all of the possibilities.

For $k' \in [1, n]$, there are $\binom{n}{k'}$ possible solutions in order to find:

$$\min_{k' \in [1,n]} \left\{ \sum_{i=1}^{n} a_i \cdot c_{il'_{nearest}} + \sum_{i=1}^{k'} c_{l'_i l'_{nearest}} \right\}$$

This method requires a time complexity of $O(2^n)$ to compute and evaluate every possible solution. The Brute-force method can guarantee a global optimum, however it is too time-consuming and not scalable to evaluate a large WMN consisting of a large number of routers. On the other hand, as mentioned in previous chapters, the equipment (mesh routers) that will be used to accommodate these algorithms has very limited computing resources. Therefore, it is necessary to investigate a heuristic approach that has higher efficiency.

Heuristic Solution

The purpose of this proposed heuristic algorithm (Algorithm 7) is to place file copies iteratively based on predicted access rates and storage spaces with the aim of minimizing the expected cost.

The algorithm takes a vector of previous copy locations L and a vector of past access records A as input, and delivers a vector of new copy locations L' as output (i.e., to place the copies) so that the total expected cost can be minimized. Intuitively, k (or $|L|$) represents the number of previous copies in

the network, n (or $|A|$) is the number of access requests for the file from the whole network, and k' (or $|L'|$) refers to the number of copies that will exist in the network after the transfer.

At the start of the procedure, the algorithm initializes two sets: L' is initially an empty set, while G' contains all of the servers in the network. The algorithm also computes the original total cost from original copy locations L. Then, the procedure goes into a loop, which performs as the main part of the algorithm. The loop will first choose the node (MeshFS server) with the greatest access density from the network, which means that this node has the largest number of access requests from both itself and its neighbours. The algorithm then calculates the total expected cost of adding this node into L'. If this cost is less than the previous cost from the last round, this node will be added into L', and it will be removed from set G' as well as its neighbours. This loop will continue until the cost can no longer be reduced. Finally, the set L' will be the output result.

Algorithm 7 Find Locations of Copies

Input: previous locations of copies L, access records A
Output: new locations of copies L'

```
1:  procedure FindLocations
2:      L' ← ∅
3:      G' ← set of servers
4:      originCost ← calCost(L)
5:      lastCost ← originCost
6:
7:      loop
8:          l ← server ∈ G' with max(accesses + ∑(neighbour))
9:          curCost ← calCost(L' + {l})
10:
11:         if curCost ≥ lastCost then
12:             break
13:         end if
14:
15:         lastCost ← curCost
16:         L' ← L' + {l}
17:         G' ← G' \ ({l} + neighbours of l)
18:     end loop
19:
20:     return L'
21: end procedure
```

The calculation of the total expected cost of a set of servers is abstracted as another procedure (Algorithm 8). As discussed in previous sections, the total

expected cost consists of two parts: the expected cost for every request server from A to download the nearest copy from (L', k'), and the cost to transfer current copies or make new copies from (L, k) to (L', k'). The accumulative result will be the total expected cost of applying new locations of copies L' into MeshFS in the WMN.

Algorithm 8 Calculate Total Expected Cost

Input: previous locations of copies L, current locations of copies L', access records A

Output: total expected cost c

```
1:  procedure FindLocations
2:      c ← 0
3:
4:      for each l' ∈ L' do
5:          l ← the closest server to l' ∈ L
6:          c ← c + transfer cost from l to l'
7:      end for
8:
9:      for each a ∈ A do
10:         l' ← the closest server to a ∈ L'
11:         c ← c + transfer cost from l' to a
12:     end for
13:
14:     return c
15: end procedure
```

This heuristic algorithm will significantly reduce the computing time needed to find a solution to the problem, however it may converge into a local rather than a global optimum. Hence, there is a tradeoff between efficiency and accuracy. The performance of this algorithm will be evaluated in the next section through simulations.

5.3 Simulation & Evaluation

In order to evaluate the performance of this algorithm for a group-based adaptive data management mechanism, a simulation program was implemented. The entire simulation procedure consists of four steps:

1. Initialize the network configuration, set simulation parameters, and generate access requests;

2. Calculate the original cost based on the network topology and vector of access requests;

3. Re-allocate file replication(s) according to the algorithm;

4. Calculate the current cost and compare it to the original cost.

In the simulations, various parameters (e.g., number of routers, bandwidth of links, number of files, file sizes, number of access requests) are varied to evaluate their effects. In the simulation, the expected access vector $\mathbf{A}$ is generated by uniform distribution.

5.3.1 Influence of a Single Parameter

The following simulations essentially study the cost reduction resulting from the application of the algorithm, in comparison with the original cost. The cost reduction reflects the cost that can be saved by applying the algorithm. The higher the cost reduction, the better the performance. In each case, only one parameter will change while other parameters stay constant. For example, if the original cost is $10c$, the new cost after applying the heuristic algorithm is c. The cost reduction will then be: cost reduction $= 1 - \frac{c}{10c} = 90\%$.

Number of Routers

In this simulation, the network bandwidth is fixed at 100 MB/s, and there are 20 files with file sizes of 512 MB. Each file has 20, 50, or 100 requests from different routers. The number of routers varies from 5 to 200.

Fig. 5.6 illustrates the relationship between cost reduction and number of routers. It can be observed that the algorithm performs best when the number of routers is relatively small (approximately 20). The cost reduction decreases when the number of routers increases. This decrease cost reduction stops after the number of routers exceeds 50. The reason for this behaviour is that due to the large scale of the network, the algorithm tends to generate more file replications to ensure a lower access cost. However, more file copies also lead to higher costs in making these copies. Hence, the cost reduction becomes stable.

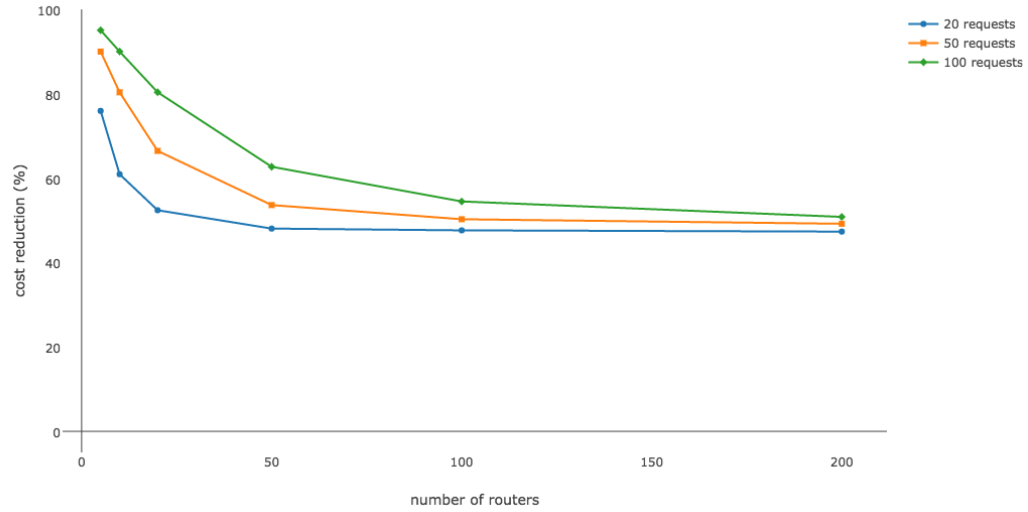

Fig. 5.6: Relationship between Cost Reduction and Number of Routers

Link Bandwidth

In this simulation, the number of routers in the network is set to be 20, 50, and 100, with bandwidth varying from 5 to 200 MB/s. The number of files is 20, with fixed file sizes of 512 MB, while each file has 20 requests from different routers.

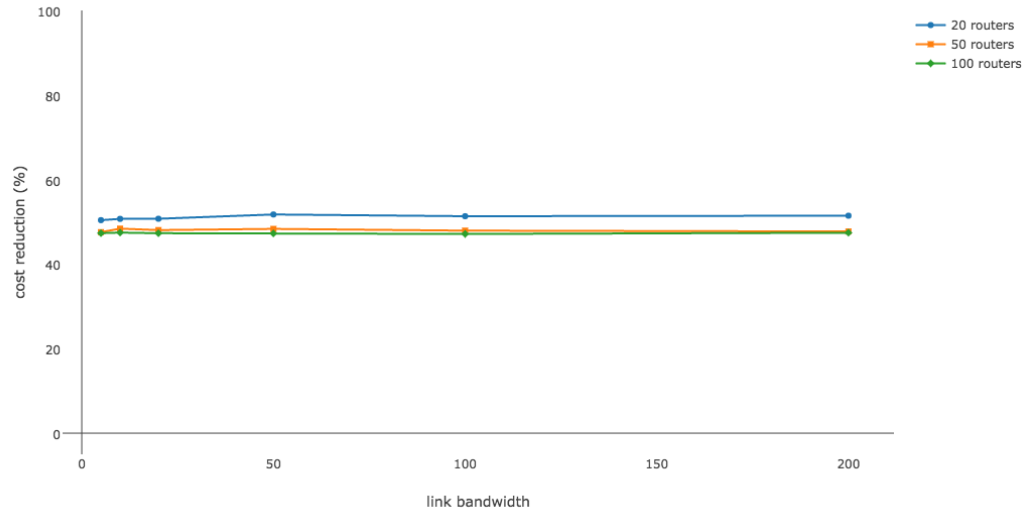

Fig. 5.7: Relationship between Cost Reduction and Link Bandwidth

Fig. 5.7 shows the relationship between cost reduction and link bandwidth in the network. It can be seen that the link bandwidth between mesh routers has no effect on the performance of this algorithm, although the algorithm will

take bandwidth into account in calculating the cost. However, the change in bandwidth will not affect the cost reduction because we are evaluating relative ratios instead of absolute costs.

Number of Files

In this simulation, the number of routers in the network is set to be 20, 50, and 100, with a bandwidth of 100 MB/s. The number of files varies from 5 to 200, with fixed file sizes of 512 MB, while each file has 20 requests from different routers.

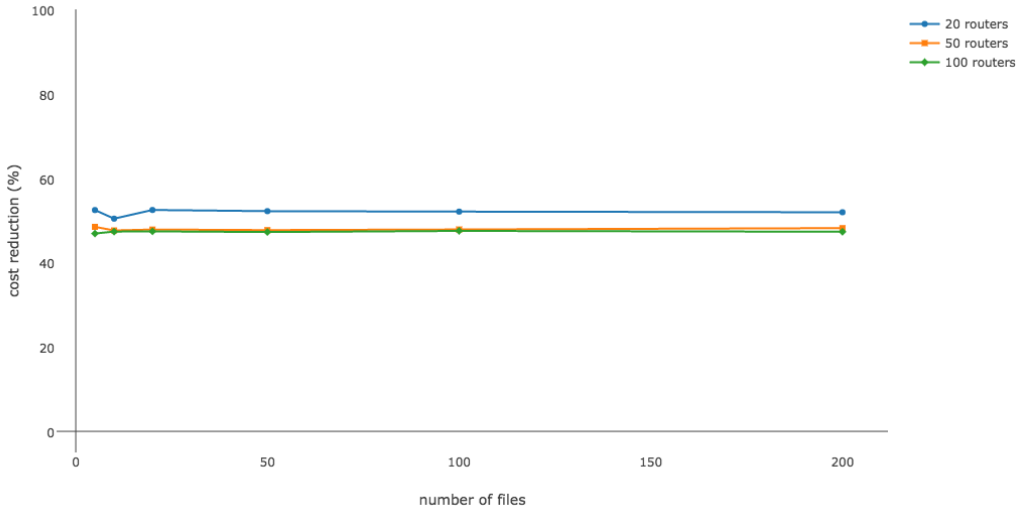

Fig. 5.8: Relationship between Cost Reduction and Number of Files

Fig. 5.8 shows the relationship between cost reduction and number of files. It can also be observed that the number of files has no influence on the algorithm's performance because the algorithm is file-oriented. For each file, the algorithm will be applied to adjust the locations of its copies. Hence, it is not concerned about how many files exist in the network.

File Size

In this simulation, the number of routers in the network is set at 20, 50, and 100, with a bandwidth of 100 MB/s. The number of files is 20, with file sizes varying from 4 MB to 2048 MB, while each file has 20 requests from different routers.

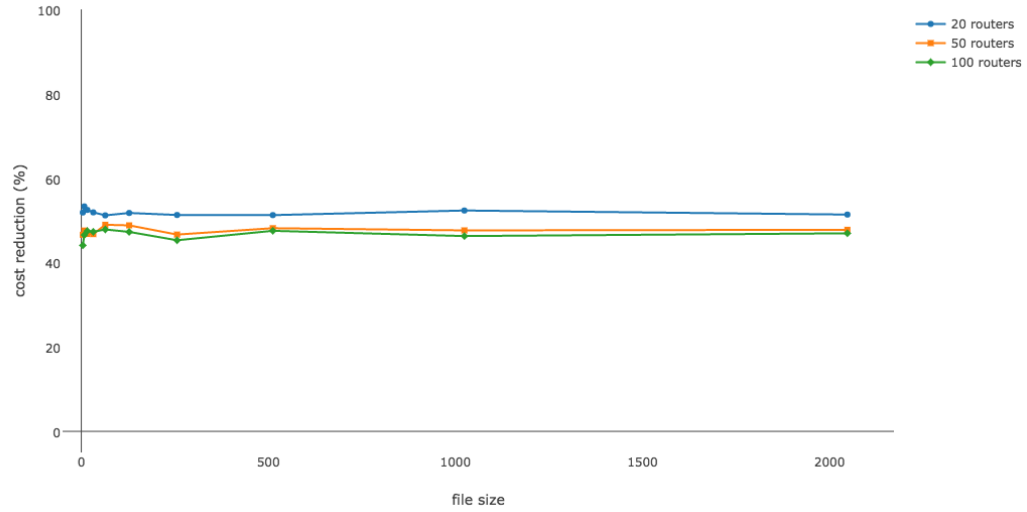

Fig. 5.9: Relationship between Cost Reduction and File Size

Fig. 5.9 illustrates the relationship between cost reduction and file size. Similar to link bandwidth, file sizes do not affect the algorithm's performance since the size is first used to calculate the cost but is later cancelled, during the calculation of the cost ratio.

Number of Requests

In this simulation, the number of routers in the network is set to be 20, 50, and 100, with a bandwidth of 100 MB/s. The number of files is 20, with fixed file sizes of 512 MB, while each file has 5 to 200 requests from different routers.

Fig. 5.10 shows the relationship between cost reduction and the number of access requests for each file. It can be observed that the cost reduction increases along with the number of requests. The reason for this has been explained in a previous e-book section: generating a copy is a one-time cost; however, if requests come from the same place, multiple costs can be saved. Hence, the algorithm will be more effective when there are more access requests.

5.3.2 Influence of a Combination of Parameters

From previous simulations, it can be concluded that only changes in the number of routers and the number of requests will affect the performance of this algorithm, while link bandwidth, number of files, and file size have no

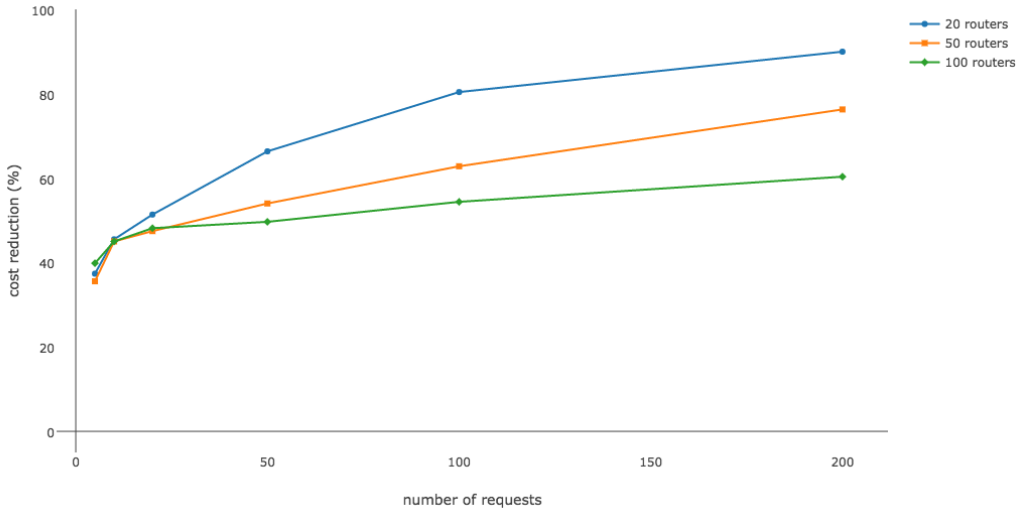

Fig. 5.10: Relationship between Cost Reduction and Number of Requests

influence on its effectiveness. Hence in this simulation, the number of routers varies from 5 to 200, with the network bandwidth fixed at 100 MB/s, the number of files at 20 with a file size of 512 MB, while each file has 5 to 200 requests from different routers.

Number of Routers vs. Number of Requests

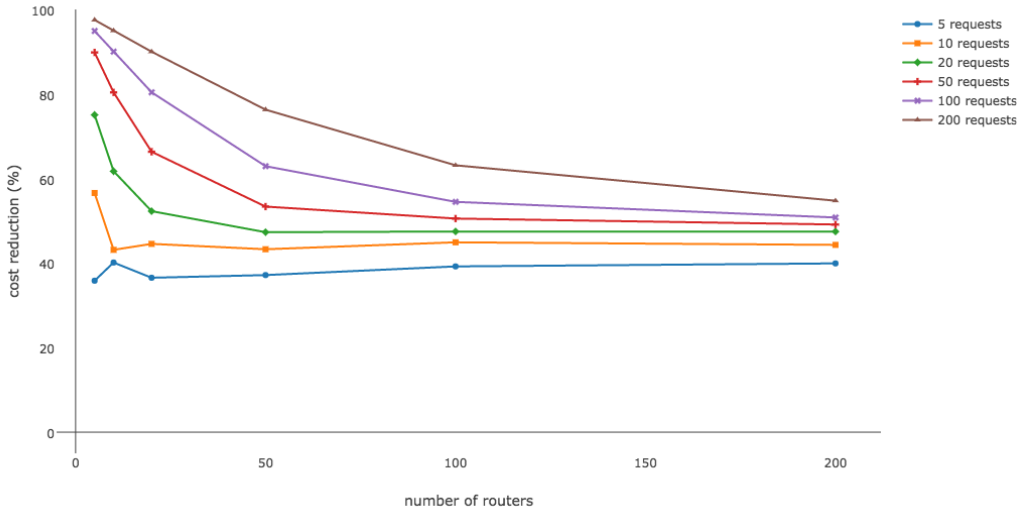

Fig. 5.11: Relationship between Cost Reduction and Number of Routers with Different Numbers of Requests

Fig. 5.11 illustrates the relationship between cost reduction and the number of routers and number of requests. It can be observed that the trends are similar: the cost reduction will decrease when the number of routers increases, and the reduction will increase if there are more access requests for each file.

Number of Requests vs. Number of Routers

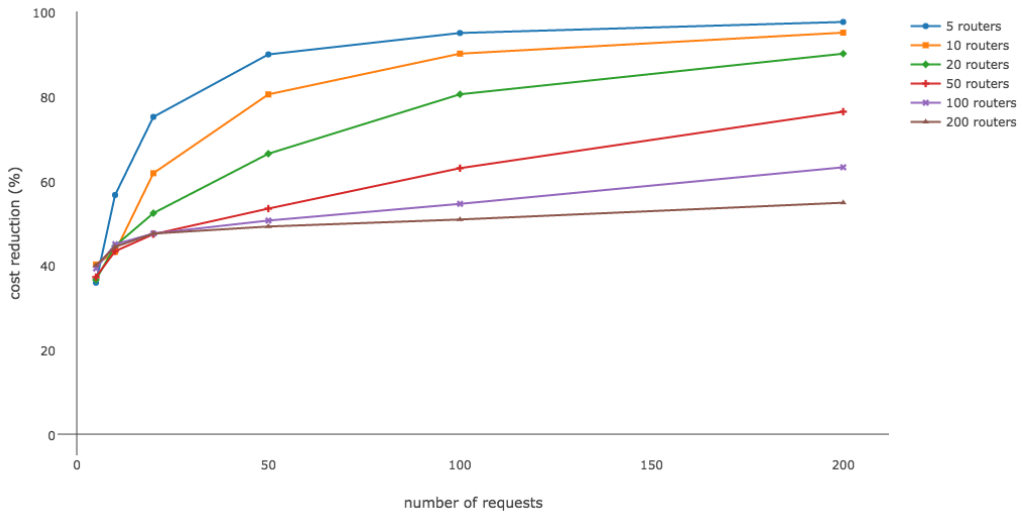

Fig. 5.12: Relationship between Cost Reduction and Number of Requests with Different Numbers of Routers

Fig. 5.12 also shows the same relationship but from a different perspective. Similar results can also be seen in this diagram: fewer routers will keep the cost reduction high, thus the performance will be better, while more requests can improve the utilization of replications, hence enhancing effectiveness.

Surface Diagram

Fig. 5.13 is a 3D surface diagram that describes the occurrence of cost reduction under different numbers of routers and numbers of requests. This surface diagram is a combination of line plots from the previous two diagrams, which presents a more comprehensive view. To facilitate the interpretation of this diagram, higher values of cost reduction are marked in red while smaller values are marked in blue.

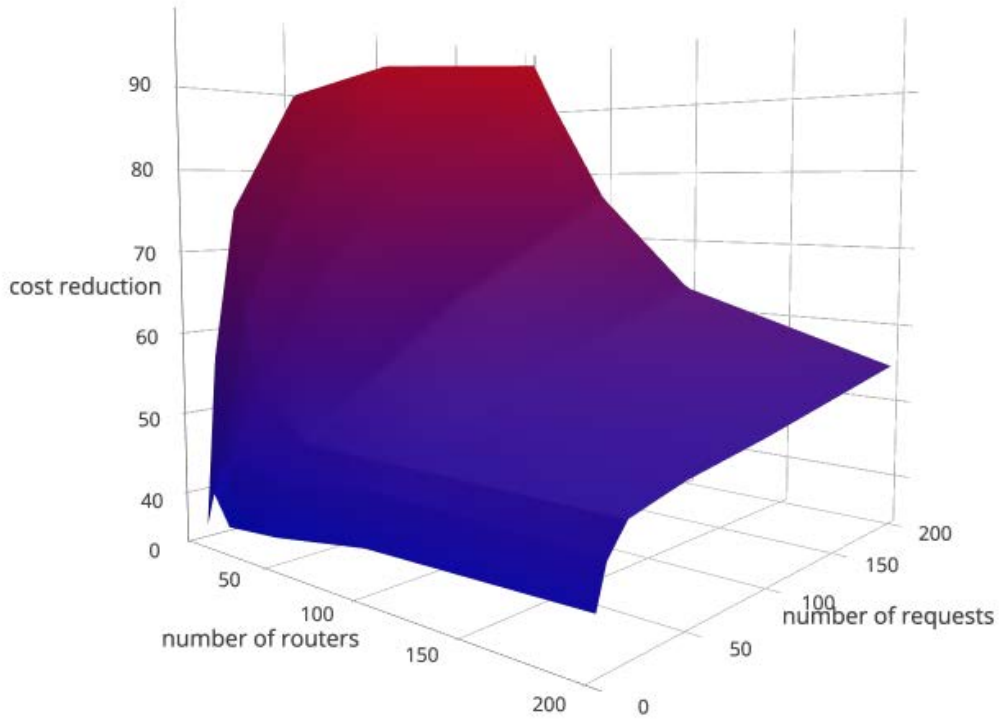

Fig. 5.13: Surface Diagram of Cost Reduction, Number of Requests, and Number of Routers

It can be observed that when there are only a few requests, the cost reduction will be low (around 35%), and the number of routers has only a small effect on the cost reduction. When the number of requests starts growing, the influence from the number of routers becomes more significant. When the scale of the network becomes smaller and the number of requests becomes larger, the cost reduction increases more rapidly, with a maximum change of nearly 97.5%.

From the diagram we can observe that the cost reduction will increase more quickly when the number of requests grows. However, when the number of routers grows at the same pace, the cost reduction will drop more slowly. Hence, the number of access requests has more influence than the number of mesh routers on the performance of this algorithm.

5.3.3 Comparison to the Brute-Force Solution

To better demonstrate the efficiency of this heuristic algorithm, we have also conducted experiments to compare this algorithm with the Brute-force method. As mentioned in subsection 5.2.3, the Brute-force method will take binomial

time ($O(2^n)$ time complexity) to compute and evaluate every possible solution to deliver a global optimum. Hence, we have two results in this comparison:

1. Cost and effectiveness ratio to evaluate the effectiveness of the heuristic algorithm as compared to the global optimum;

2. Computation time and efficiency ratio to compare the time consumed by these two algorithms.

In this simulation, the number of routers in the network varies from 5 to 15, with a bandwidth of 100 MB/s. The number of files is 5, with fixed file sizes of 512 MB, while each file has 5 to 200 requests from different routers.

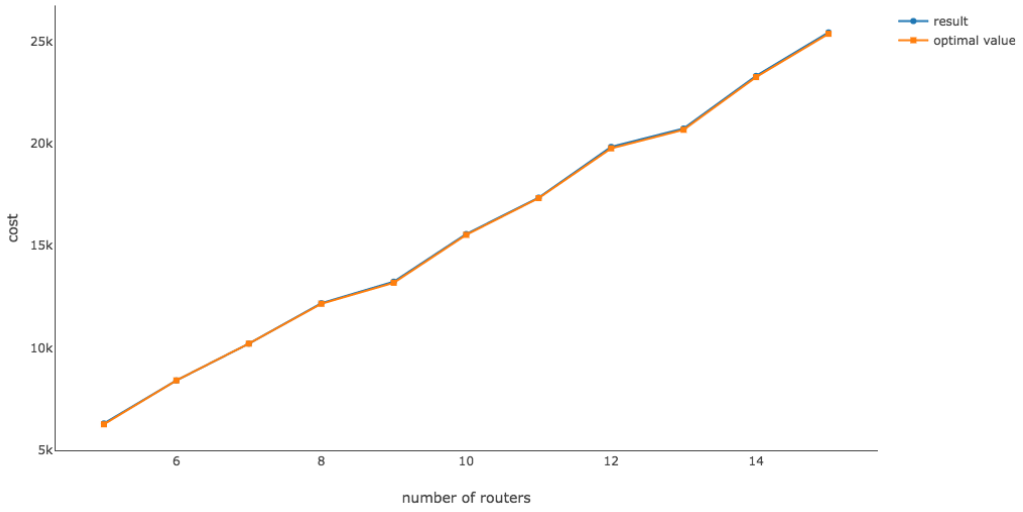

Fig. 5.14: Relationship between Cost and Number of Routers in Comparison with the Brute-Force Algorithm

Fig. 5.14 describes the relationship between the recorded cost and the number of routers. The blue line shows the result of the heuristic algorithm, while the orange line shows the optimal value of the Brute-force method. It can be observed that these two lines almost coincide with each other, which means that the results from the heuristic algorithm are very close to the optimal result.

Fig. 5.15 shows the relationship between the effectiveness ratio and the number of routers. The line represents the average values, while the scattered dots are recorded values. The effectiveness ratio gives the ratio of the cost reductions of the heuristic algorithm and the Brute-force algorithm. As previously discussed, the heuristic algorithm might deliver a local optimum,

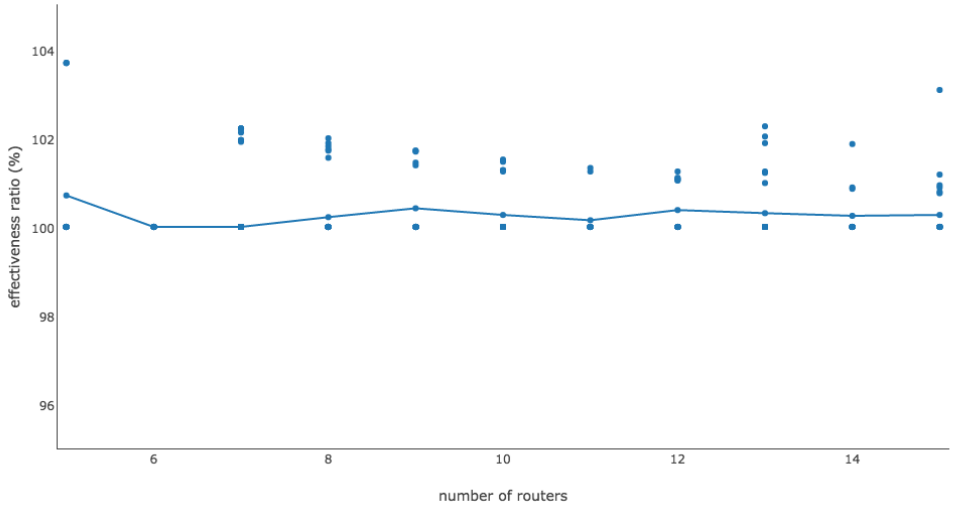

Fig. 5.15: Relationship between the Effectiveness Ratio and the Number of Routers in Comparison with the Brute-Force Algorithm

while the Brute-force algorithm can guarantee the global optimum at the expense of a long computation time. Hence, if the effectiveness ratio is equal to 100%, this means that the heuristic algorithm has reached the global optimum. If the ratio is greater than 100%, this means that the heuristic algorithm only gives out a local optimum as a result. From Fig. 5.15 we see that most of the time, this heuristic algorithm can achieve optimal results. In addition, the results are generally close to the global optimum, even when the heuristic algorithm cannot reach the global optimum. The difference is only within 4%, thus indicating the effectiveness of the heuristic algorithm.

Fig. 5.16 shows the relationship between computation time and the number of routers. The blue line shows the time taken by the heuristic algorithm, while the orange line shows the time taken by the Brute-force method. It can be observed that the time taken by the Brute-force method is growing exponentially, while the time taken by the heuristic algorithm stays at a very small value (less than one second).

Fig. 5.17 shows the relationship between the efficiency ratio and the number of routers. The line was drawn along the average values, while the scattered dots are recorded values. The efficiency ratio compares the heuristic algorithm and the Brute-force algorithm in the time needed to deliver the results. As previously mentioned, the Brute-force method takes a time complexity of $O(2^n)$, where n represents the number of routers in the network. Hence, the time needed to compute the result through the Brute-force method increases

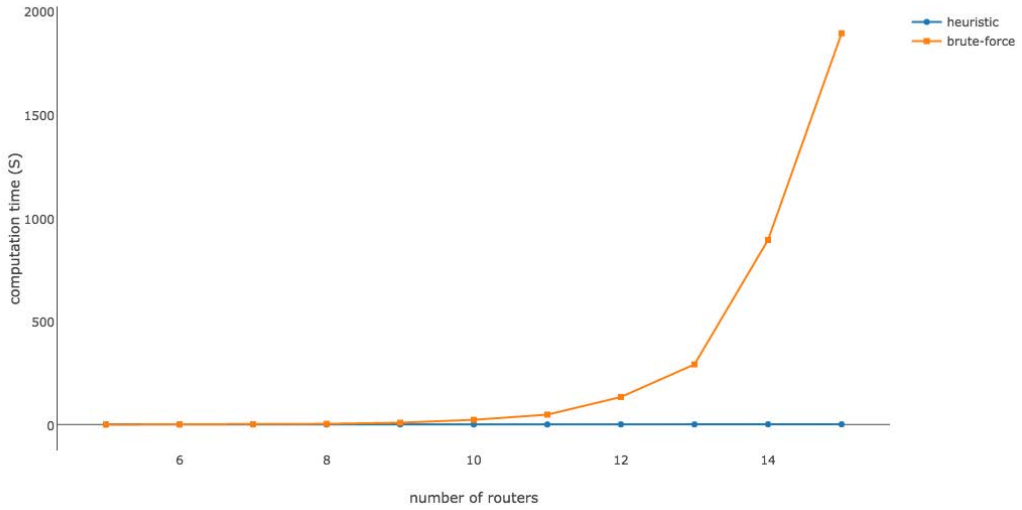

Fig. 5.16: Relationship between Computation Time and Number of Routers in Comparison with the Brute-Force Algorithm

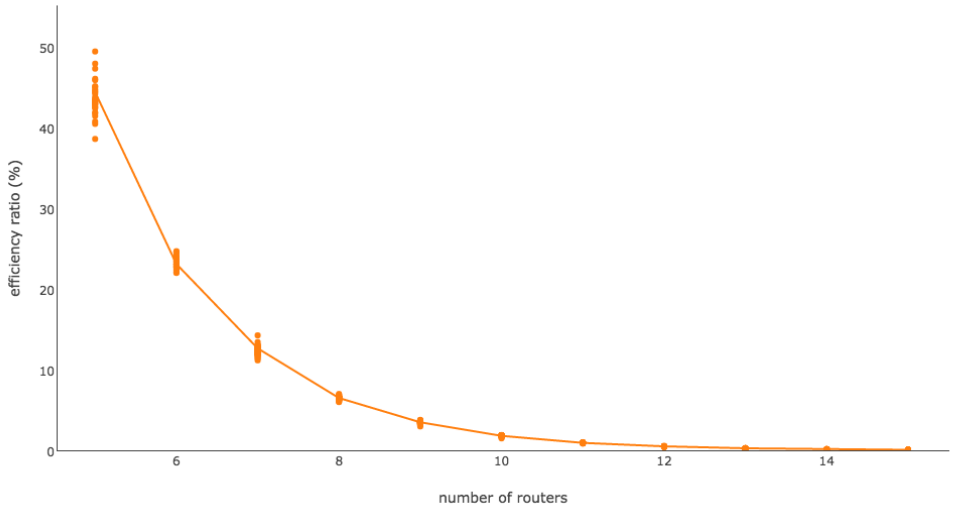

Fig. 5.17: Relationship between the Efficiency Ratio and the Number of Routers in Comparison with the Brute-Force Algorithm

exponentially, and Fig. 5.17 indicates that the ratio curve drops exponentially. When the number of routers in the network is 5, the heuristic algorithm will take about 45% to 50% of the amount of time used by the brute-force method to output the result. When there are eight routers in the network, the ratio drops to within 10%. After the size of the network exceeds 10, the heuristic algorithm needs only less than 1% of the amount of time to deliver the result as compared to the time required to try out all possibilities. Again, the results indicate that the heuristic algorithm should be effective.

5.3.4 Comparison to Genetic Algorithm

Model

In order to apply the genetic algorithm to the group-based adaptive data management mechanism, the mathematical models need to be adjusted to suit the features of the genetic algorithm. In this experiment, we involved available memory spaces on mesh routers to consider multiple files at one time. In this GA model, as with the heuristic algorithm, there are also n servers and m files. The n servers are denoted as $\mathbf{R} = [r_1, r_2, \dots r_n]$ with memory space $\mathbf{M} = [m_1, m_2, \dots m_n]$; the m files are denoted as $\mathbf{F} = [f_1, f_2, \dots f_m]$ with size $\mathbf{S} = [s_1, s_2, \dots s_m]$.

This time we use a matrix to represent the locations of copies:

$$\mathbf{L} = \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & l_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{m1} & l_{m2} & \cdots & l_{mn} \end{bmatrix}$$

where each l_{ij} indicates the existence of file i at router j. l_{ij} can be either 0 or 1. 1 means that the router has a copy of this file; 0 otherwise. The matrix $\mathbf{L}$ can be used in the genetic algorithm to represent chromosomes. The validity of a chromosome can be verified by: $\mathbf{SL} \leq \mathbf{M}$, which is

$$\begin{pmatrix} s_1 & s_2 & \cdots & s_m \end{pmatrix} \begin{pmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & l_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{m1} & l_{m2} & \cdots & l_{mn} \end{pmatrix} \leq \begin{pmatrix} m_1 & m_2 & \cdots & m_n \end{pmatrix}$$

If $\mathbf{SL}$ is less than $\mathbf{M}$, this chromosome $\mathbf{L}$ is valid.

Algorithm

1. Initialization
   The original copy locations L, which is the first place where the file has been submitted, will be initialized as the first generation of the

genetic algorithm. Inspired by [24, 25, 26], a structure of 2-dimensional chromosome is adopted to better illustrate the relationship between copy location, copy size, and the available space of routers. The structure and usage of this kind of 2-dimensional chromosome will be explained in the crossover section.

2. Evaluation
   The same procedure used in the heuristic algorithm to compute the cost after applying L is used as a fitness function to evaluate the value of this chromosome (shown in Algorithm 9).

Algorithm 9 Calculate the Total Expected Cost

Input: previous locations of copies L, current locations of copies L', access records A
Output: total expected cost c

```
1:  procedure FindLocations
2:      c ← 0
3:
4:      for each l' ∈ L' do
5:          l ← the closest server to l' ∈ L
6:          c ← c + transfer cost from l to l'
7:      end for
8:
9:      for each a ∈ A do
10:         l' ← the closest server to a ∈ L'
11:         c ← c + transfer cost from l' to a
12:     end for
13:
14:     return c
15: end procedure
```

The total expected cost consists of two parts: the expected cost for every request server from A to download the nearest copy from (L', k'), and the cost to transfer the current copies or to make new copies from (L, k) to (L', k'). The accumulative result will be the total expected cost of applying new locations of copies L' into MeshFS in the WMN.

3. Selection
   As mentioned earlier, a chromosome can be verified by $\mathbf{SL} \leq \mathbf{M}$ (i.e., to test whether it is a valid chromosome). If the result of multiplying **SL** is less than or equal to **M**, which means that every mesh router in the WMN has enough space to store those copies whose locations are

represented by **L**, then this chromosome is valid and will be shortlisted for reproducing the next generation. If the result of multiplying **SL** exceeds the storage capacity represented by **M**, this means the WMN does not have enough space to accommodate these copies, and thus this chromosome will be discarded.

4. Crossover
   Since the chromosomes used in this genetic algorithm are in the form of a two-dimensional matrix, the crossover procedure can be done either horizontally or vertically. The position of the crossing can also be altered in different rows/columns. In every generation, each individual will crossover with every other sibling to reproduce the next generation. The following illustrates this procedure:

   Vertical Crossover:

$$\left[\begin{array}{cc|cc} l_{11} & l_{12} & \cdots & \cdots \\ l_{21} & l_{22} & \cdots & \cdots \\ l_{31} & l_{32} & \cdots & \cdots \\ l_{41} & l_{42} & \cdots & \cdots \end{array}\right] + \left[\begin{array}{cc|cc} \cdots & \cdots & l_{13} & l_{14} \\ \cdots & \cdots & l_{23} & l_{24} \\ \cdots & \cdots & l_{33} & l_{34} \\ \cdots & \cdots & l_{43} & l_{44} \end{array}\right] \Longrightarrow \begin{bmatrix} l_{11} & l_{12} & l_{13} & l_{14} \\ l_{21} & l_{22} & l_{23} & l_{24} \\ l_{31} & l_{32} & l_{33} & l_{34} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix}$$

   Horizontal Crossover:

$$\left[\begin{array}{cccc} l_{11} & l_{12} & l_{13} & l_{14} \\ l_{21} & l_{22} & l_{23} & l_{24} \\ \hline \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{array}\right] + \left[\begin{array}{cccc} \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \hline l_{31} & l_{32} & l_{33} & l_{34} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{array}\right] \Longrightarrow \begin{bmatrix} l_{11} & l_{12} & l_{13} & l_{14} \\ l_{21} & l_{22} & l_{23} & l_{24} \\ l_{31} & l_{32} & l_{33} & l_{34} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix}$$

5. Mutation
   In addition to crossovers, mutations are another important element in genetic algorithms. A mutation makes it possible to stand out from the local optimum. It is used to introduce more possibilities into the next generation. After a crossover, each element in the chromosome matrix will change from 0 to 1 or 1 to 0 with a mutation probability of 10%.

6. Termination
   The iteration will stop when the fitness value stops increasing for several generations. One hundred generations is considered to be a suitable stopping point in this simulation because most cases become stable at this point from the results.

Example

For example, if there is a network with three routers and two files, file 1 has one copy on router 1 and router 2, and file 2 has one copy on router 1 and router 3, the matrix of the copy locations will be:

$$\mathbf{L} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

The first generation of the population will be generated by mutating this chromosome. In these experiments, each generation will keep a certain number of the best individuals to reproduce the next generation by means of crossover. For example, if we keep the best four individuals in every generation, then the first generation might be:

$$\mathbf{L1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L2} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L3} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{L4} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Children reproduced by vertical crossover before mutation:

$$\mathbf{L1} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L2} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L3} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L4} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$

$$\mathbf{L5} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L6} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L7} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{L8} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

$$\mathbf{L9} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{L10} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{L11} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{L12} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Children reproduced by horizontal crossover before mutation:

$$\mathbf{L1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{L3} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{L4} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix},$$

$$\mathbf{L5} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{L6} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{L7} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L8} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$

$$\mathbf{L9} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{L10} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L11} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{L12} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

During the crossover procedure, each individual in the current generation will conduct both a horizontal and a vertical crossover with every other sibling to generate the next population. If we need to keep the best n individuals for every generation, then there will be $P(n, 2)$ possibilities to go through. In this case, there are $P(4, 2) = 12$ children generated, and the best four will be retained for reproducing the next generation of the population. The genetic algorithm will keep iterating this procedure until no better individual can be generated in 100 rounds.

Simulation Results

In this simulation, the number of routers in the network varies from 5 to 20, with a bandwidth of 100 MB/s. The number of files is 5, with fixed file sizes of 512 MB, while each file has 5 to 200 requests from different routers.

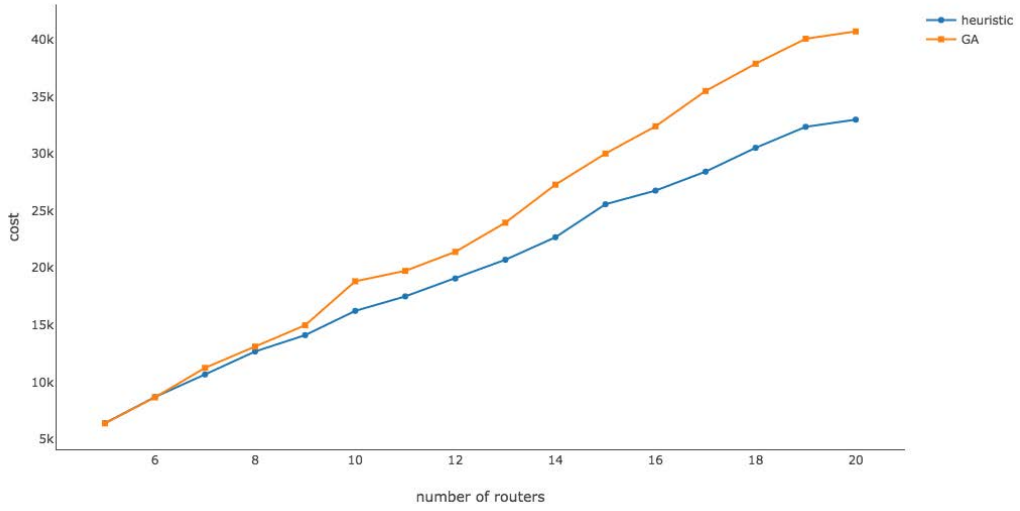

Fig. 5.18: Relationship between Cost and Number of Routers in Comparison with the Genetic Algorithm

Fig. 5.18 shows the relationship between the recorded cost and the number of routers. The blue line shows the results of the heuristic algorithm, while the orange line shows the results of the genetic algorithm. It can be observed that the heuristic algorithm has lower costs than the genetic algorithm. At the start, the results are close to each other when there are not many routers in the network. However, the difference becomes greater as the network size increases.

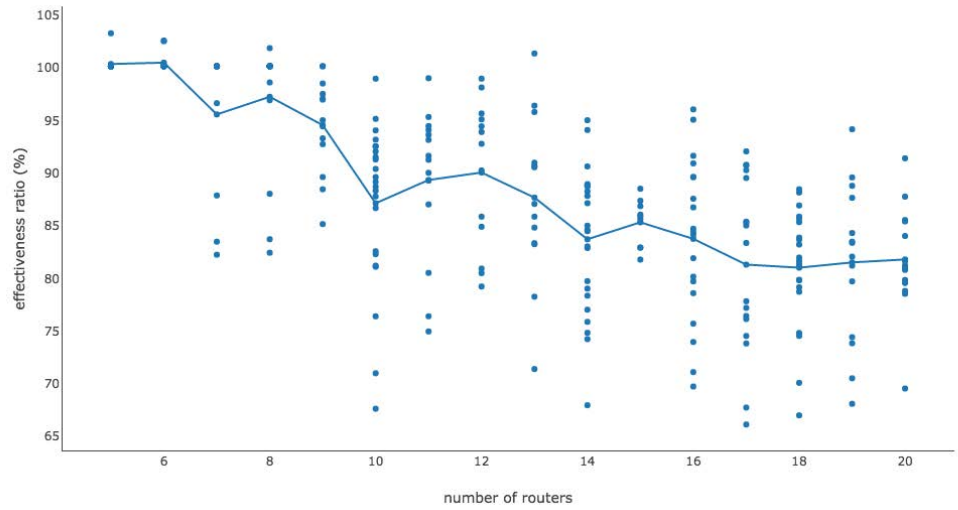

Fig. 5.19: Relationship between the Effectiveness Ratio and the Number of Routers in Comparison with the Genetic Algorithm

Fig. 5.19 shows the relationship between the effectiveness ratio and the number of routers. The line was drawn along the average values, while the scattered dots are recorded values. The effectiveness ratio reflects the comparison of the final results of the heuristic algorithm and the genetic algorithm. Both the heuristic algorithm and the genetic algorithm may only deliver a local optimum instead of the global optimum. From the scattered dots, it can be observed that in some cases, the genetic algorithm may generate less of a cost than the heuristic algorithm (ratio $> 100\%$). However, in most cases, the values of the scattered dots are less than 100%, which means that the heuristic algorithm performs better than the genetic algorithm. The average line also proves that when the network size grows, the performance of the genetic algorithm will be weaker than before.

Fig. 5.20 shows the relationship between the recorded cost and number of requests. The blue line shows the results of the heuristic algorithm, while the orange line shows the results of the genetic algorithm. It can be observed that the heuristic algorithm has lower costs than the genetic algorithm. At the start, the results are close to each other when there are not many requests for each file in the network. However, the difference becomes greater as the number of requests increases.

Fig. 5.21 shows the relationship between the effectiveness ratio and the number of requests. The line was drawn along the average values, while the scattered dots are recorded values. The effectiveness ratio reflects the comparison of the

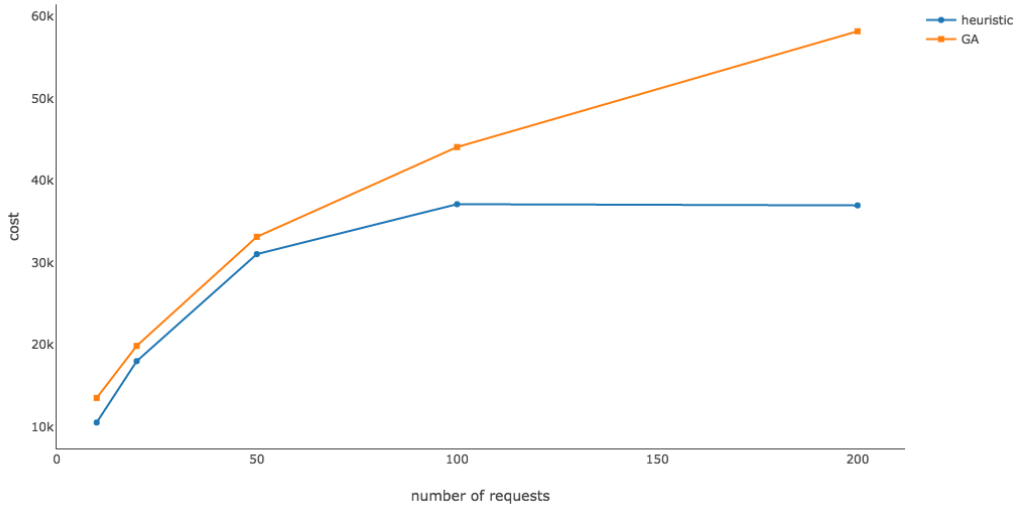

Fig. 5.20: Relationship between Cost and Number of Requests in Comparison with the Genetic Algorithm

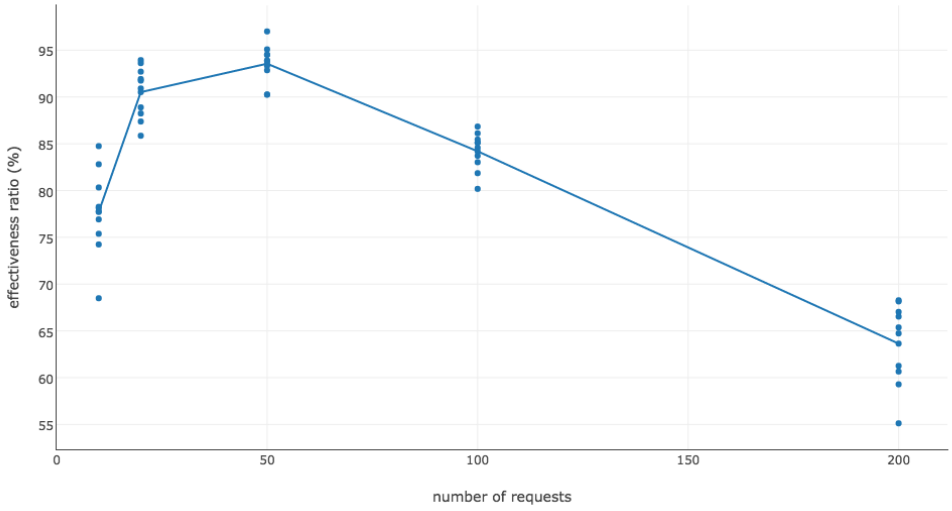

Fig. 5.21: Relationship between the Effectiveness Ratio and the Number of Requests in Comparison with the Genetic Algorithm

final results of the heuristic algorithm and the genetic algorithm. It can be observed that when the number of requests approaches 50, the performance of the GA is close to that of the heuristic algorithm. However, in most cases, the heuristic algorithm performs better than the GA because the ratio is always under 100%.

Fig. 5.22 shows the relationship between the computation time and the number of routers. The blue line shows the time taken by the heuristic algorithm,

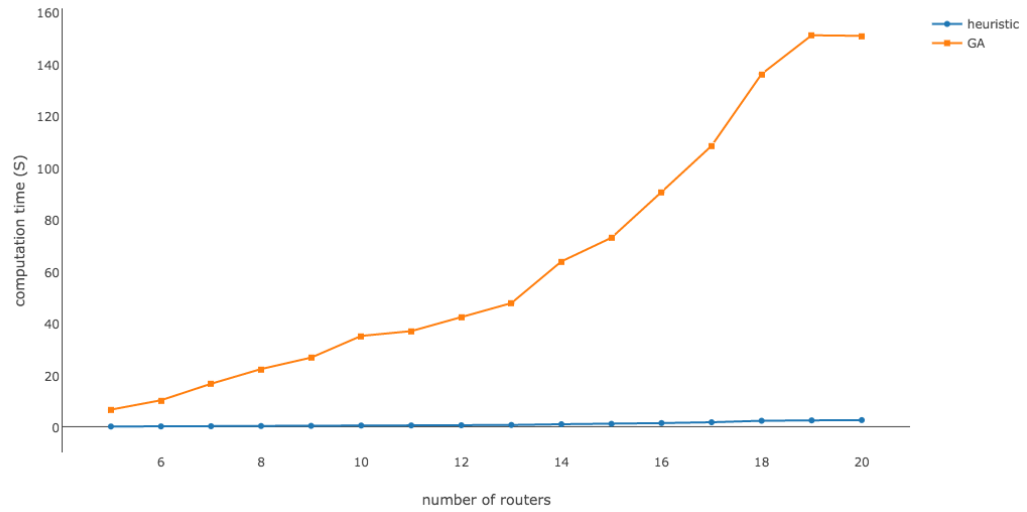

Fig. 5.22: Relationship between Computation Time and Number of Routers in Comparison with the Genetic Algorithm

while the orange line shows time taken by the genetic algorithm. It can be observed that the time taken by the genetic algorithm grows along with the increase in network size, while the time taken by the heuristic algorithm stays very small (less than 1 second).

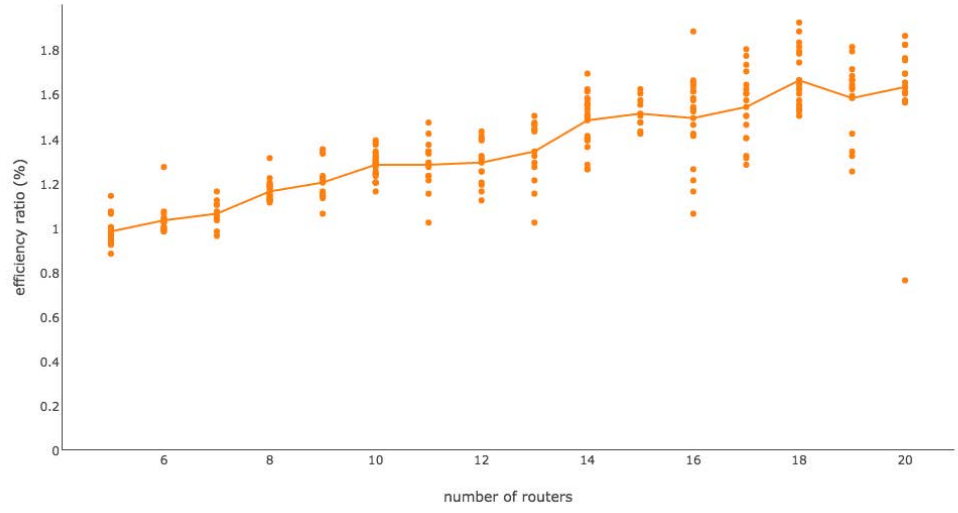

Fig. 5.23: Relationship between the Efficiency Ratio and the Number of Routers in Comparison with the Genetic Algorithm

Fig. 5.23 shows the relationship between the efficiency ratio and the number of routers. The line was drawn along the average values, while the scattered dots are recorded values. The efficiency ratio compares the heuristic algorithm with the genetic algorithm in the time needed to generate the results. As shown

in Fig. 5.22, the computation time needed by the genetic algorithm grows almost linearly along with the number of routers in the network, while the time needed by the heuristic algorithm is relatively stable. However, from this figure, it can be observed that the efficiency ratio is also growing along with the size of the network. The reason for this is that although the computation time taken by the heuristic algorithm is very small, it still grows with the size of the network. The heuristic algorithm only needs about 1% to 1.5% of the time to deliver the result. Therefore, we can draw the conclusion that the heuristic algorithm is better than the genetic algorithm in both the aspects of saving cost and saving time.

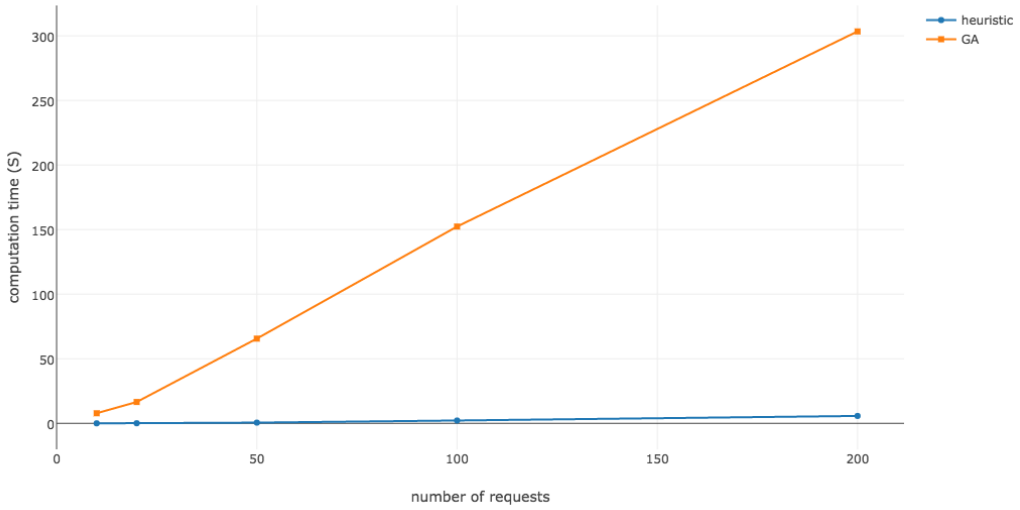

Fig. 5.24: Relationship between Computation Time and Number of Requests in Comparison with the Genetic Algorithm

Fig. 5.24 shows the relationship between the computation time and the number of requests. The blue line shows the time taken by the heuristic algorithm, while the orange line shows the time taken by the genetic algorithm. It can be observed that the time taken by the genetic algorithm grows linearly with the increase in the number of requests for each file, while the time taken by the heuristic algorithm remains very small (less than 1 second).

Fig. 5.25 shows the relationship between the efficiency ratio and the number of requests. The line was drawn along the average values, while the scattered dots are recorded values. The efficiency ratio compares the heuristic algorithm with the genetic algorithm in the time needed to generate the results. Similar to the comparison with the number of routers, the heuristic algorithm only needs about 0.2% to 2% of time required by the genetic algorithm to generate

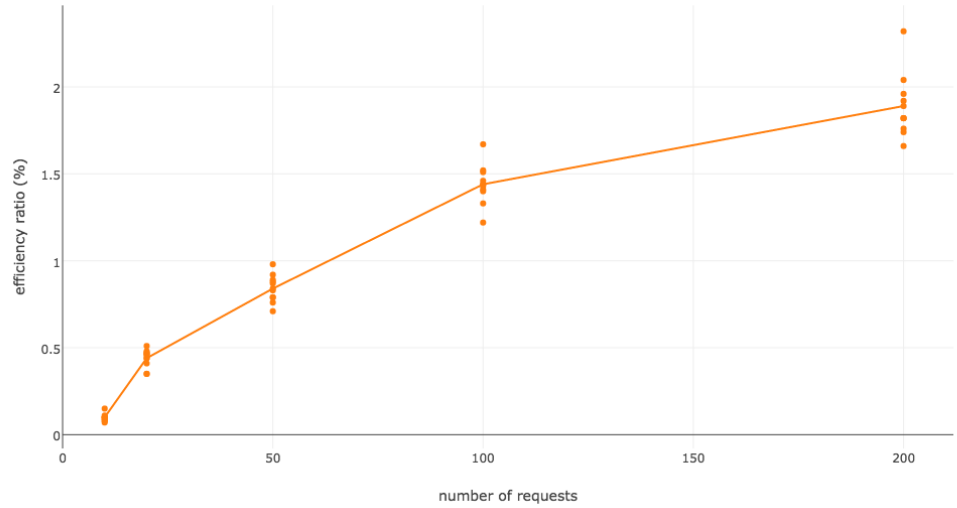

Fig. 5.25: Relationship between the Efficiency Ratio and the Number of Requests in Comparison with the Genetic Algorithm

the results. Therefore, the heuristic algorithm is better than genetic algorithm in both the aspects of saving cost and saving time.

5.4 Conclusion

In this paper, we have investigated and presented a group-based adaptive data management mechanism for cloud-based wireless mesh networks. The problem was defined and stated, and a mathematical model was established to study the problem. An algorithm was designed to compute the number of copies needed in the network and the locations to accommodate those copies in order to achieve the optimized cost. The algorithm was also evaluated by means of simulations to study its performance and effectiveness under different circumstances with different parameters.

From the simulations, we found that the algorithm will be affected by two factors: the number of routers in the network and the number of requests for access to files. The algorithm will be at its most effective when there are more access requests and fewer mesh routers in the network. To evaluate its performance in comparison with other algorithms, control experiments were conducted with the Brute-force algorithm and the genetic algorithm. The Brute-force method can guarantee the delivery of the global optimum, however the computation time required will grow exponentially. Thus, this method

is extremely slow. The genetic algorithm has the ability to stand out from a local optimum. However, the computation time cannot be predicted and the performance is not stable.

Compared to these two methods, the heuristic algorithm can generate results that are very close to the optimal value in a more efficient way, thus it should be more suitable for mesh routers. The results of the simulation show that this algorithm can produce cost savings of at least 35% compared to placing the file in the network. The derivation from the globally optimal value is generally within 4%.

Conclusion & Future Work

6

In this research project, we have first reviewed and studied the background and existing works for wireless mesh networks, cloud computing, and distributed file systems. Inspired by these techniques, we have designed a cloud-based wireless mesh network equipped with cloud-like features so that clients can upload/manage their personal data onto the network. Next, we have investigated and implemented a person-based data management mechanism that allows the network to dynamically move personal data along with clients' movement to enhance access efficiency.

In order to better enhance flexibility and configurability of the data management mechanisms, we have developed MeshFS - a light weight, fault-tolerant distributed file system specifically designed for wireless mesh networks. Based on MeshFS, we have investigated a group-based data management mechanism applicable to scenarios in which multiple clients need to share/access the same file. This mechanism includes a heuristic algorithm that can intelligently replicate and re-allocate data copies to mesh routers to optimize access costs. Compared to statically hosted data uploaded onto a cloud-based network, this mechanism can save at least 35% of costs after re-allocation. The results are very close to the global optimum and can be achieved in a significantly efficient amount of time.

These proposed mechanisms and systems can significantly enhance the functions of wireless mesh networks, especially in public or educational environments. Compared to mainframe servers, mesh routers have only limited computing and storage resources, however routers are much less expensive and can easily be installed around the world. The value of this research project is to integrate and enhance scattered resources from the network in order to achieve higher computing power and provide more advanced services. In the future, this can be further expanded to introduce more advanced features into cloud-based wireless mesh networks, such as data version control, video streaming, pattern analysis & recognition, etc. The ultimate goal is to form a large WMN system with delimited cloud features.

References

[1] Ian F Akyildiz and Xudong Wang. "A survey on wireless mesh networks". In: *Communications Magazine, IEEE* 43.9 (2005), S23–S30 (cit. on p. 5).

[2] Ian F Akyildiz, Xudong Wang, and Weilin Wang. "Wireless mesh networks: a survey". In: *Computer networks* 47.4 (2005), pp. 445–487 (cit. on p. 5).

[3] Michael Armbrust, Armando Fox, Rean Griffith, et al. "A view of cloud computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58 (cit. on p. 9).

[4] Djohara Benyamina, Abdelhakim Hafid, and Michel Gendreau. "Wireless mesh networks design—A survey". In: *Communications Surveys & Tutorials, IEEE* 14.2 (2012), pp. 299–310 (cit. on p. 5).

[5] Dhruba Borthakur. "The hadoop distributed file system: Architecture and design". In: *Hadoop Project Website* 11.2007 (2007), p. 21 (cit. on p. 12).

[6] Peter J Braam et al. *The Lustre storage architecture*. 2004 (cit. on p. 12).

[7] Tracy Camp, Jeff Boleng, and Vanessa Davies. "A survey of mobility models for ad hoc network research". In: *Wireless communications and mobile computing* 2.5 (2002), pp. 483–502 (cit. on p. 6).

[8] Shin-Ming Cheng, Phone Lin, Di-Wei Huang, and Shun-Ren Yang. "A study on distributed/centralized scheduling for wireless mesh network". In: *Proceedings of the 2006 international conference on Wireless communications and mobile computing*. ACM. 2006, pp. 599–604 (cit. on pp. 6, 15).

[9] Chi-Yin Chow, Hong Va Leong, and Alvin TS Chan. "Distributed group-based cooperative caching in a mobile broadcast environment". In: *Proceedings of the 6th international conference on Mobile data management*. ACM. 2005, pp. 97–106 (cit. on p. 6).

[10] Chi-Yin Chow, Hong Va Leong, and Alvin TS Chan. "GroCoca: group-based peer-to-peer cooperative caching in mobile environment". In: *Selected Areas in Communications, IEEE Journal on* 25.1 (2007), pp. 179–191 (cit. on p. 6).

[11] Chi-Yin Chow, Hong Va Leong, and Alvin TS Chan. "Group-based cooperative cache management for mobile clients in a mobile environment". In: *Parallel Processing, 2004. ICPP 2004. International Conference on*. IEEE. 2004, pp. 83–90 (cit. on p. 6).

[12] Saumitra M Das, Himabindu Pucha, and Y Charlie Hu. "Hop-by-Hop Cooperative Caching in Wireless Mesh Networks: A Design and Implementation Study". In: (2006) (cit. on pp. 6, 15).

[13] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on p. 12).

[14] Mieso K Denko, Thabo Nkwe, and Mohammad S Obaidat. "Efficient cooperative caching with improved performance in wireless mesh networks". In: *Communications (ICC), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1–5 (cit. on pp. 7, 15).

[15] Benjamin Depardon, Gaël Le Mahec, and Cyril Séguin. "Analysis of six distributed file systems". In: (2013) (cit. on p. 13).

[16] Alexandros G Dimakis, P Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. "Network coding for distributed storage systems". In: *Information Theory, IEEE Transactions on* 56.9 (2010), pp. 4539–4551 (cit. on p. 13).

[17] Richard Draves, Jitendra Padhye, and Brian Zill. "Routing in multi-radio, multi-hop wireless mesh networks". In: *Proceedings of the 10th annual international conference on Mobile computing and networking*. ACM. 2004, pp. 114–128 (cit. on pp. 6, 15).

[18] Florian Fainelli. "The OpenWrt embedded development framework". In: *Proceedings of the Free and Open Source Software Developers European Meeting*. 2008 (cit. on pp. 18, 33).

[19] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system". In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM. 2003, pp. 29–43 (cit. on p. 12).

[20] "Gluster". In: *http:/www.gluster.org* (2008) (cit. on p. 12).

[21] Csaba Henk and Miklos Szeredi. "FUSE: Filesystem in Userspace". In: *Online at http://sourceforge.net/projects/fuse* 92 (2012) (cit. on p. 37).

[22] Pieter Hintjens. "ZeroMQ: Messaging for Many Applications". In: (2013) (cit. on p. 36).

[23] Ivan Wang-Hei Ho, Patrick P Lam, Peter Han Joo Chong, and Soung Chang Liew. "Harnessing the High Bandwidth of Multiradio Multichannel 802.11n Mesh Networks". In: *Mobile Computing, IEEE Transactions on* 13.2 (2014), pp. 448–456 (cit. on p. 8).

[24] Chuyen Khoa Huynh and Won Cheol Lee. "An interference avoidance method using two dimensional genetic algorithm for multicarrier communication systems". In: *Communications and Networks, Journal of* 15.5 (2013), pp. 486–495 (cit. on p. 83).

[25] Chuyen Khoa Huynh and Won Cheol Lee. "Multicarrier cognitive radio system configuration based on interference analysis by two dimensional genetic algorithm". In: *Advanced Technologies for Communications (ATC), 2011 International Conference on*. IEEE. 2011, pp. 85–88 (cit. on p. 83).

[26] Chuyen Khoa Huynh and Won Cheol Lee. "Two-dimensional genetic algorithm for OFDM-based cognitive radio systems". In: *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. IEEE. 2011, pp. 100–105 (cit. on p. 83).

[27] Tomasz Imieĺinski and BR Badrinath. "Data management for mobile computing". In: *ACM Sigmod Record* 22.1 (1993), pp. 34–39 (cit. on p. 6).

[28] Tomasz Imielinski and BR Badrinath. "Mobile wireless computing: challenges in data management". In: *Communications of the ACM* 37.10 (1994), pp. 18–28 (cit. on p. 6).

[29] Shudong Jin and Limin Wang. "Content and service replication strategies in multi-hop wireless mesh networks". In: *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. ACM. 2005, pp. 79–86 (cit. on p. 6).

[30] Jangeun Jun and Mihail L Sichitiu. "The nominal capacity of wireless mesh networks". In: *Wireless Communications, IEEE* 10.5 (2003), pp. 8–14 (cit. on pp. 6, 15).

[31] "Laguna GW2388-4 Single Board Computer". In: *http://www.gateworks.com/-product/item/laguna-gw2388-4-network-processor* (2015) (cit. on pp. 18, 33).

[32] Peter Mell and Tim Grance. "The NIST definition of cloud computing". In: (2011) (cit. on p. 9).

[33] Stanislav Miskovic and Edward W Knightly. "Routing primitives for wireless mesh networks: Design, analysis and experiments". In: *INFOCOM, 2010 Proceedings IEEE* (2010), pp. 1–9 (cit. on pp. 6, 15).

[34] "MooseFS". In: *http://www.moosefs.org/* (2015) (cit. on p. 12).

[35] Alok Nandan, Shirshanka Das, Giovanni Pau, Mario Gerla, and MY Sanadidi. "Co-operative downloading in vehicular ad-hoc wireless networks". In: *Wireless On-demand Network Systems and Services, 2005. WONS 2005. Second Annual Conference on*. IEEE. 2005, pp. 32–41 (cit. on p. 6).

[36] Vishnu Navda, Anand Kashyap, and Samir R Das. "Design and evaluation of imesh: an infrastructure-mode wireless mesh network". In: *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a*. IEEE. 2005, pp. 164–170 (cit. on pp. 6, 15).

[37] Eldad Perahia and Robert Stacey. "Next Generation Wireless LANS: 802.11n and 802.11ac". In: (2013) (cit. on p. 6).

[38] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. "Cloud computing: an overview". In: *Cloud Computing*. Springer, 2009, pp. 626–631 (cit. on p. 10).

[39] Ashish Raniwala and Tzi-cker Chiueh. "Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network". In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. IEEE. 2005, pp. 2223–2234 (cit. on p. 5).

[40] Mahadev Satyanarayanan. "A survey of distributed file systems". In: *Annual Review of Computer Science* 4.1 (1990), pp. 73–104 (cit. on p. 12).

[41] Mahadev Satyanarayanan, James J Kistler, Puneet Kumar, et al. "Coda: A highly available file system for a distributed workstation environment". In: *Computers, IEEE Transactions on* 39.4 (1990), pp. 447–459 (cit. on p. 12).

[42] Sayandeep Sen and Bhaskaran Raman. "Long distance wireless mesh network planning: problem formulation and solution". In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 893–902 (cit. on pp. 6, 15).

[43] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system". In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE. 2010, pp. 1–10 (cit. on p. 12).

[44] Lisa Ward. "802.11ac Technology Introduction". In: *Rohde & Schwarz White Paper* (2012) (cit. on p. 6).

[45] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. "Ceph: A scalable, high-performance distributed file system". In: *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, pp. 307–320 (cit. on p. 12).

[46] Weigang Wu and Yifei Huang. "Hierarchical Cooperative Data Caching for Wireless Mesh Networks". In: *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*. IEEE. 2010, pp. 289–296 (cit. on pp. 7, 15).

[47] Wenzheng Xu, Weigang Wu, Hejun Wu, and Jiannong Cao. "A Cooperative Approach to Cache Consistency Maintenance in Wireless Mesh Networks". In: *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*. IEEE. 2011, pp. 512–519 (cit. on pp. 7, 15).

[48] S. T. Yang, Henry C. B. Chan, Patrick P. Lam, and Peter H. J. Chong. "A Cloud-based Wireless Mesh Network with Adaptive Data Storage Functions". In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2. 2015, pp. 540–545 (cit. on p. 33).

[49] Jing Zhao, Ping Zhang, Guohong Cao, and Chita R Das. "Cooperative caching in wireless p2p networks: Design, implementation, and evaluation". In: *Parallel and Distributed Systems, IEEE Transactions on* 21.2 (2010), pp. 229–241 (cit. on p. 7).

[50] Jing Zhao, Ping Zhang, and Guohong Cao. "On cooperative caching in wireless P2P networks". In: *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*. IEEE. 2008, pp. 731–739 (cit. on p. 7).