International Journal of Hybrid Information Technology
Vol.8, No.3 (2015), pp.395-406
http://dx.doi.org/10.14257/ijhit.2015.8.3.36

# Recovering YAFFS2 Files for Forensic Analysis Based on Timestamps and Tnode Trees

Na Huang[1], Jingsha He[1,2], Bin Zhao[1], Xuejiao Wan[1], Gongzheng Liu[1] and Mingming Duan[2]

[1]*School of Software Engineering*
*Beijing University of Technology, Beijing 100124, China*
[2]*Information Technology Department*
*Beijing Development Area Co., Ltd., Beijing 100176, China*
*E-mail:15264712494@163.com*

***Abstract***

*As Android-based intelligent devices get more popular, digital technologies for forensic investigation have received increasingly more attention. Among the main technical issues in digital forensics, however, data recovery requires a significant amount of effort. In this paper, we first analyze the characteristics of the NAND flash storage as well as the mechanisms in the YAFFS2 file system. We then propose a file reconstruction method based on timestamps using Tnode trees in the YAFFS2 file system. Based on the last access timestamp information in the object header and the process of creating Tnode tree, the proposed method can be used to locate valid data blocks so as to recover the original files and would thus be able to reconstruct the file system. Experiments conducted under the Linux operating system over image files show that the proposed method could recover the final version of files effectively and would also perform more efficiently compared to similar methods.*

***Keywords:*** *Security, Digital Forensics, Data Recovery, YAFFS, Timestamp, Tnode*

## 1. Introduction

With the rapid development of mobile communications technologies, information has become an indispensable resource in our everyday life. However, such communication technologies raise various security issues while providing a great deal of convenience to users. One such great challenge is that more and more criminal cases and civil disputes now involve electronic data. Consequently, there is the need to prove the existence of some events in a way that can be admitted into the court through forensic approaches. Since mobile devices, especially smart phones and intelligent terminals with multiple functions and capabilities, are widely used, they have drawn a great deal of attention from security engineers as well as from forensic investigators.

As an operating system designed specifically for mobile devices, Android has so far taken about 80% of market share in the worldwide and, without much doubt, has become a major platform for acquiring and analyzing digital evidence. During the process of digital forensics, one clear observation is that deleted data on smart phones is possibly related to user behavior and could be an important part of the digital evidence. As a result, recovering data from static storage media in a forensically sound manner is necessary during any investigation procedure. Data recovery from Android-based intelligent devices has to involve NAND, the non-volatile flash memory which has been widely used in Android smart phones, and YAFFS2 (Yet Another Flash File System 2) which is the file system that has been designed specifically for the NAND flash memory that Google has

ISSN: 1738-9968 IJHIT

adopted as the official file system for Android smart phones and other types of intelligent devices.

In this paper, we first explore the characteristics of NAND flash memory and YAFFS2 file system. To reduce the time it takes to reconstruct files, we take advantage of the Tnode tree structure to build file system so that each data page only needs to be scanned once in the recovery process. In addition, since different versions of files can be distinguished by timestamps, we only reconstruct the latest version of the file system based on the time of the last access to files. As the result, we propose a new method to recover files from YAFFS2 based on timestamps using the Tnode tree. We will also perform some experiments to show that the proposed method is effective and can reconstruct files more efficiently.

The rest of this paper is organized as follows. In Section 2, we review some related work. In Section 3, we describe some main characteristics of NAND and YAFFS2 for data recovery. In Section 4, we describe our proposed method which includes a flow diagram and a step-by-step description of the method. In Section 5, we present some experiment results to show the performance advantages of the proposed method. Finally, in Section 6, we conclude this paper in which we also discuss our plan for future work.

## 2. Related Work

There has been some research effort on data recovery that relies on the metadata of the underlying file system. Luck and Stokes proposed an approach to recover files from handset memory dumps after studying the structure of the FAT file system [1], which afterwards was also applied to the Ext3 and Ext4 file systems. Manning described a reasonable amount of core mechanisms that make YAFFS work including coding strategies, garbage collection and object management [2], which helps greatly in the understanding of YAFFS. In [3], a data recovery program was proposed for Android-based mobile phones that was claimed to be able to recover over 90% of modified files without using a backup strategy, which shows that data that has not yet been erased by the garbage collection could be recovered effectively. Although Android phones can serve as a large data repository that would allow forensic engineers to acquire relevant data, collect information about their owners as well as a variety of facts that are under investigation, however, specific features of each smartphone platform generally have to be considered prior to data acquisition. Paper [4] proposed a method to perform data acquisition from Android smartphones regardless of versions and manufacturers. Yang, *et al.*, performed some experiment with the Linux operating system to demonstrate that using reversed-scanning and YAFFS2 metadata, deleted files could be correctly recovered [5]. The main metadata that was used in the experiment includes object ID, chunk ID and sequence number of blocks. Spreitzenbarth, *et al.*, provided an overview of the YAFFS2 file system from the point of view of digital forensics and demonstrated how garbage collection and wear leveling techniques could affect the recoverability of deleted and modified files [6]. Sack, *et al.*, analyzed the structure of the Android system and created a forensic guide with the conclusion that all data stored on Android smart phones could be examined [7]. Sylve, *et al.*, discussed some of the challenges in performing Android memory acquisition and developed a new kernel module called *dmd* for memory dumping [8]. In the work, the kernel structure was analyzed through using newly developed volatility functionality and the results illustrated the great potential that deep memory analysis could offer to digital forensic investigators. It was claimed that if spare area of flash memory pages doesn't exist or it is created from the unallocated area of the undamaged file system, reconstruction of the file system would not be possible [9]. To solve the problem of fragmentation, the work also proposed some new analysis techniques for fragmented flash memory pages in smartphones. Dibb and Hammoudeh developed an

open-source toolkit that is aimed to automatically extract and handle all data from Android-based devices [10]. Wu et al. presented a recovery approach for SQLite history recorders from YAFFS2 that can correctly recover updated or deleted records in Android smart phones [11] through acquiring Android image in logical method and then recording the object ID, object type and chunk ID of each chunk sequentially to prepare for reversed-scanning. During the scanning process, useful information is stored in an array and the timeline of SQLite operating log is constructed. Based on previous studies [5, 11], they further proposed a metadata-based method for recovering file traces that contain all the versions of files from YAFFS2 [12].

The above effort shows that it is possible to recovery files from NAND flash memory as long as the actual data still resides on the flash chip. But all of their method are attempt to find all pages belong to the current file according to object ID when meet an object header. It may results that scanning a data page two or more times.

## 3. YAFFS and Tnode Tree

### 3.1 The NAND Flash

NAND is an electrically erasable and reprogrammable non-volatile flash memory with such advantages as low-power, portable and low cost. NAND must be erased first before it can be rewritten, which is very different from the magnetic storage medium. The NAND flash memory is organized into a series of blocks with 32, 64 or 128 pages in each block. The basic unit of writing operation for NAND flash is one page while data has to be erased in blocks [6]. A page is further divided into usable and spare areas. The usable area is used to store user data while the spare area is used to store file system metadata [5, 12]. There is a limit on the number of times that NAND flash blocks can be reliably programmed and erased. A technique known as wear ensures that all physical blocks are exercised uniformly. To maximize the life of NAND, it is critical to implement both wear leveling and garbage collection. An effective strategy is out-of-place updates in which new data cannot be overwritten at the same location occupied by old data [6]. This strategy is forensic-friendly since deleted files still exist in NAND and can thus be recovered during investigation [6].

To reflect the special requirements of NAND, the design of YAFFS2 implements wear-leveling, bad block management, chip erase operation and address distribution using Meta tag information. YAFFS2 creates its own data structures on the chip where the file system is loaded and the data is loaded into memory of the host as response.

### 3.2 The YAFFS2 File System

YAFFS (Yet Another Flash File System) is an open-source file system that is designed to be fast, robust and suitable for embedded use with NAND and NOR Flash. YAFFS has been widely used in Linux in consumer devices. YAFFS2 is the second version of YAFFS evolved from YAFFS1 to accommodate newer chips that supports both 2K-byte and 512-byte page flash while YAFFS1 only supports the original 512-byte page flash.

**3.2.1 Main Characteristics of YAFFS2**：The YAFFS2 file system takes the advantage of NAND by making a page as the minimum unit for write operation. In YAFFS2, files are stored as either a file object header or as a data type in fixed-size chunks in the NAND flash memory. There are thus two types of chunks: data chunks and object header chunks [10]. Data chunks contain the actual file content while object header chunks contain file/directory metadata and descriptor information, such as file size, object name and creation time [12]. Each chunk has Tags in the spare area that holds additional

information such as the chunk ID, serial number, number of bytes and object ID. Files stored on the NAND chips have unique identities, *i.e.*, the object IDs, since they are regarded as objects by the file system. When a file is modified, the file system will store the new data in empty pages and add a new object header [12]. When a file is deleted, YAFFS2 will write a new object header at the end of the file to indicate that the file has been deleted. In order to read only the actual object headers, YAFFS2 marks every newly written block with a sequence number that is monotonically increasing and all pages in the same block share this sequence number. The sequence of the chunks can be inferred from the block sequence number and the chunk offset within the block. Therefore, when YAFFS2 scans the flash and detects multiple chunks that have an identical object ID and chunk ID, it can choose which to use by taking the largest sequence number [6]. As far as YAFFS2 is concerned, only one block is considered the allocating block at a time. The file system will look for another empty block as the next allocating block as soon as the current block is fully allocated. Because it allows the creation of a chronologically ordered list, the first occurrence of any object ID and chunk ID is the most active occurrence.

**3.2.2 Garbage Collection in YAFFS2**：As we can see so far, when a block is made up of only deleted chunks, that block can be erased and reused [5]. However, YAFFS2 will invoke the garbage collection process only when most of the flash chip is used and there is data waiting to be written [2]. It will select blocks where whole or most chunks are out of data, copy the new data to a clean area, delete the original data and make those blocks as being erased. Then, those erased blocks can be reused like new blocks. When garbage collection starts, YAFFS2 will no longer allocate pages in the physical order, thus the block sequence numbers will play an important role in determining the allocation order of blocks so as to tell which chunks are the latest [2]. Therefore, YAFFS2 will never rewrite the original chunks until there are no sufficient blocks for allocation. This forensically friendly strategy is called “out-place-write”. As a result of the “out-place-write” strategy and wear-leveling principle, a file on the NAND flash is going to be divided into several sections when either add or modify operations occur. However, division occurs only in the boundary of NAND pages. After garbage collection, there will have file fragmentations in the NAND flash due to the impact of division as well as garbage collection strategy. Some part of a file may get erased while some other parts may still be stored in different blocks or different pages within the same block.

**3.2.3 Involved YAFFS2 Metadata**：Some useful information for recovery that is stored in the object header chunks are displayed in Table 1 according to YAFFS2 documentation and our own research [5] among which *Yst_ctime* indicates when the file is created, *Yst_mtime* indicates the time when this file is last modified and *Yst_atime* indicates the time of the last access to this file which will changes as the read operation done.

**Table 1. Object Header Information Used in This Paper**

| Content | Size(Bytes) | Comment |
|---|---|---|
| Yaffs_ objectType | 4 | File/directory/links |
| Parent_ Objectid | 4 | Parent directory |
| Name[yaffs_max_name_length+1] | 256 | File name |
| Yst_mod | 4 | Permissions |
| Yst_uid | 4 | User_ID |
| Group_id | 4 | Group_ID |
| Yst_atime | 4 | Access time |
| Yst_mtime | 4 | Modified time |

| | | |
|---|---|---|
| Yst_ctime | 4 | Create time |
| Filesize | 4 | Length of the file |
| isShrink | 4 | Whether the file is resized |

**Table 2. Information in the Tags**

| Content | Size(Bits) | Comment |
|---|---|---|
| Chunk ID | 20 | Sequence in the file |
| Object ID | 18 | Unique identifies of file |
| Serial Number | 2 | Identify chunks with the same chunk ID |

There are also some important tags information stored in data chunks [2]: object ID indicates which object file the chunk belongs to, so if its value is 0 then this is not part of any object; chunk ID tells the logical sequence of this chunk in the file to which it belongs, so if its value is 0 then this is a header chunk, else it is a data chunk. In YAFFS2, each chunk is marked with a 2-bit serial number that is incremented every time a chunk with the same object ID and chunk ID is replaced. This number is used to identify whether a chunk is valid. When a chunk is updated, the serial number will incremented. Information in the tags is listed in Table 2.

### 3.3 Tnode Tree

For YAFFS2, the content of each and every file is stored in chunks and these chunks are organized into a Tnode tree for the purpose of easy access. A Tnode tree provides a mapping to physical chunk address from file address [2]. Every object holds a Tnode tree. YAFFS2 Tnode tree is built with different pointers. Tnodes at the lowest level, *i.e.*, Tnodes at level 0, has 16 physical chunk indexes that point to the chunk's logical location in memory [2]. Every Tnode has 8 pointers to point to the other nodes at the next level. We can use *YAFFS_FindLevel0Tnode()* to find the position of a physical chunk [2]. Figure 1 illustrates the structure of Tnode tree in YAFFS2 in which physical chunk indexes indicate the chunk's location on the NAND flash. The tree must get updated whenever chunks get updated.

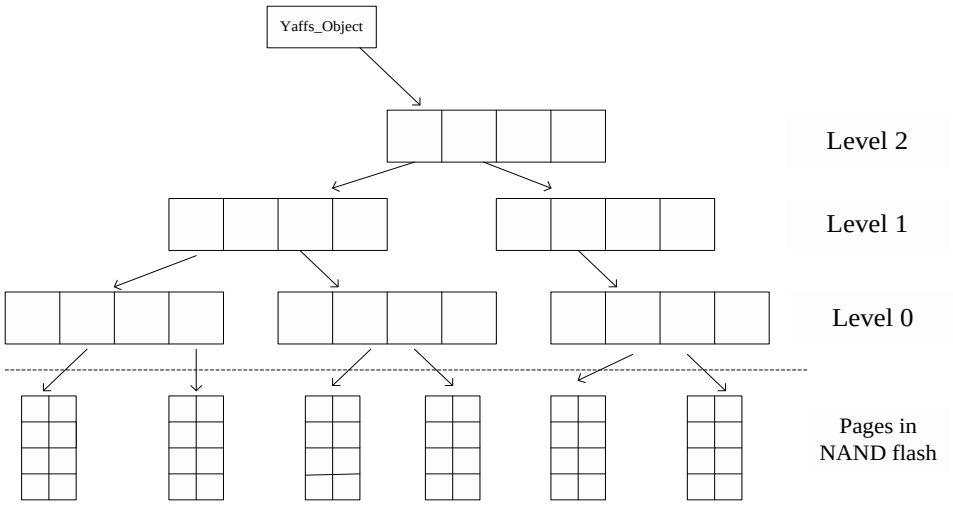



**Figure 1. The Structure of Tnode Tree**

# 4. A New Method for File Recovery

## 4.1 File Versions

As we described in Section 2, all the chunks that belong to the same file share the same object ID. If a file is modified, the *Yst_mtime* of this file will be change and the *Yst_ctime* of the newer version of chunks must be later than that of the old ones since the new chunks are always allocated after the old ones. The object header with the latest *Yst_atime* must be the final version of a file.

## 4.2 The Proposed Method

When YAFFS2 sets up the structure of file system in memory, it needs to scan partitions first. Let $Block_N$ denote the block with sequence number N and $Page_M$ denote the Mth page in the block. We begin with the block with the largest sequence number and scan the whole block from the last chunk to the first one. The general steps of the scanning algorithm are as follows:

(1) Scan $Block_N$;
(2) Scan $Page_M$ in $Block_N$. If $Page_M$ is the first one in the block, jump to (4);
(3) M=M-1, go to (2);
(4) If N=0, then end; else N=N-1, return to (1).

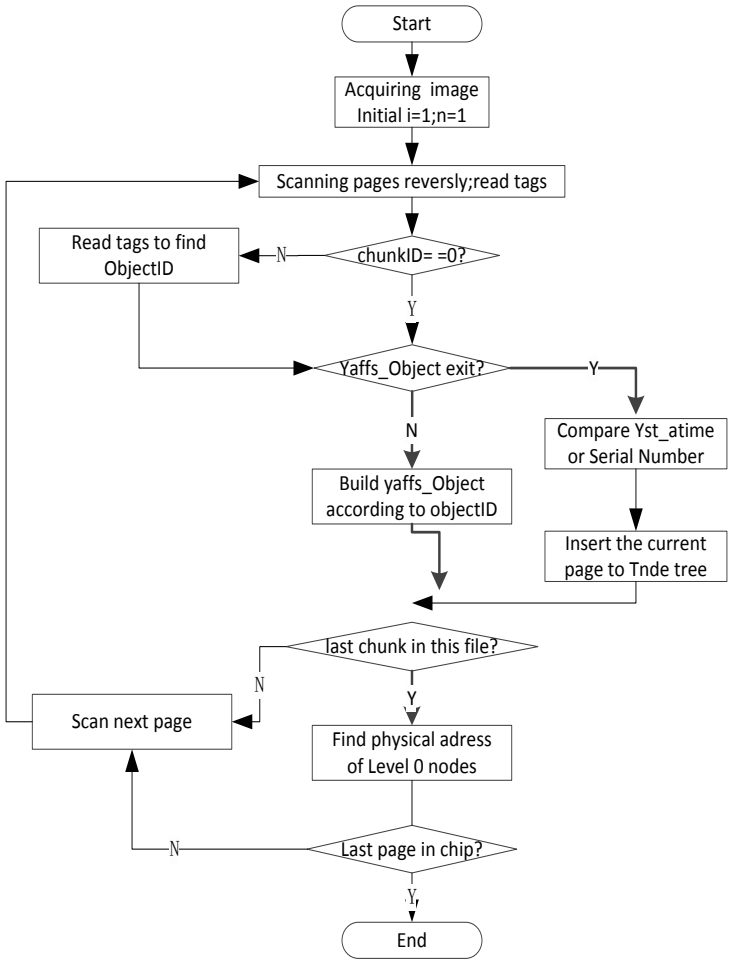



**Figure 2. The Procedure of File Recovery**

During the scanning, we take different measures to deal with situations when chunkID=0 and when chunkID>0. We can use *yaffs_FindChunkInGroup()* and *yaffs_FindLevel0Tnode()* to find the physical address of the pages at Level 0. We then restore all the chunks and reconstruct them. In our algorithm, we define a two-dimensional array File[i][j] to store the physical address of pages in which index j is the chunk ID and index i represents different files. Figure 2 illustrates the procedure of file recovery while the steps of the recovery algorithm are presented below:

(1) Initialization n=1;
(2) Scan NAND flash using the scanning algorithm above; read the yaffs_tags from the spare area of each page in the NAND flash;
(3) Build the relevant Tnode tree for every file in which there are two different circumstances as follows:
chunkID=0:
- If the yaffs_Object has been built in the RAM already, we only need to compare their Yst_atime and take the one with the latest Yst_atime;
- Else build a yaffs_Object according to its objectID;

chunkID>0:
- If there exists an object header that has the same objectID, insert the present chunk into its yaffs_Object;
- Else if such an object header doesn't exist yet, build a yaffs_Object according to the objectID and insert the current chunk;

- Else if the chunk that shares the same chunkID has been added into the relevant file's Tnode tree already, determine which one is valid based on the Serial Number and abandon the invalid one;

(4) Once the construction of the Tnode tree of one file is completed, call yaffs_FindChunkInGroup() and yaffs_FindLevel0Tnode() to find the physical address of Level 0 pages and store them in File[n][i] where i= chunkID;
(5) Copy pages into the new area according to the physical address stored in File[n][i];
(6) If the current page is the last one in this NAND flash, then end scanning; else n=n+1, go to (2).

### 4.3 Analysis of Time Saving

The proposed method for recovering files and reconstructing the file system in YAFFS2 intends to find all the pages that belong to the same file according to objectID in an object header. The storage mechanism of YAFFS2 is designed to store new data in empty pages and to add a new object header when a file is modified or deleted. Fragmentation would then occur if files are modified as illustrated in Figure 3. The scanning algorithm used would get to object header of FileX first and would then go on scanning the pages in the forward direction. It will scan pages located between chunks of this file, read their tags and abandon them instead of just skipping them. After the current file is recovered completely, it will go back to the page where fragmentation has occurred. In our proposed method, we only scan each page in the NAND flash once in the reverse sequence. Let $T_r$ denote the time that is needed to read tags in one page. Assuming that there are n pages embedded in chunks that belong to FileX, to recovery FileX, the time saving is thus $n*T_r$.

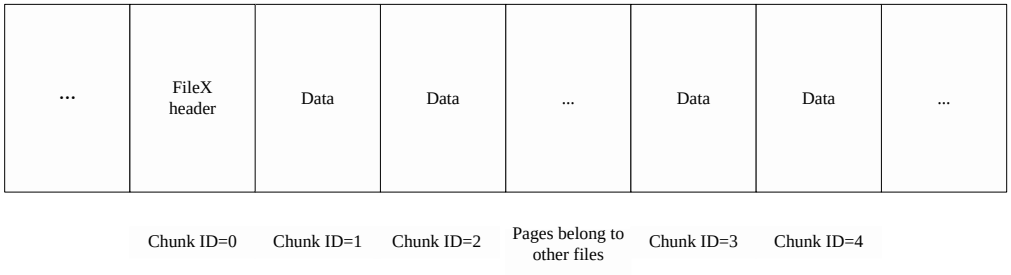



**Figure 3. Fragmentation in File X**

## 5. Experiments and Analysis

We have performed some experiments to evaluate the proposed file recovery method. The first experiment that we performed is on the recovery of files from an image file on a simulated NAND flash and the second one is on the recovery of files from eight image files. While the first experiment aims to show the effectiveness of the proposed method, the second one is used to demonstrate that the proposed method can achieve better performance over existing methods.

### 5.1 Evaluation on Effectiveness

We conducted this experiment over a VirtualBox that was set up with the Ubuntu operating system. Since the Ubuntu Linux kernel version 2.6.28-11-generic does not include a YAFFS2 file system, we need to mount YAFFS2 using the NAND flash simulator *nandsim*.

To setup up the simulation environment, the YAFFS2 source code package *yaffs2.tar.gz* must be downloaded from http://www.alphl.co.uk/cgi-bin/viewvc.cgi/yaffs2/tar/gz?view=tar [11] followed with decompressing the source code

package and compiling it. We simulated a MTD in the random access memory RAM to mount YAFFS2 partitions and created a simulation NAND flash using *nandsim*. The storage capacity of NAND flash that we have simulated is 64MB and the size of each page is 2KB. We would then be able to correctly mount the YAFFS2 file system. Afterwards, we write a file into the simulated NAND flash and then delete it after having modified it twice. The main command sequence used is listed below:

```
curl http://www.alphl.co.uk/cgi-bin/viewvc.cgi/yaffs2/tar/gz?view=tar>yaffs2.tar.gz
tar xzf yaffs2.tar.gz;make;
modprobe mtd;
modprobe mtdblock;
modprobe nandsim frist_id_byte=0x20 second_id_byte=0xa2;
third_id_byte=0x00 fourth_id_byte=0x15;
mount –t yaffs2 dev/mtdblock0  ~mnt/affs2/;
mount | grep yaffs2;
nano -w ~/mnt/yaffs2/test.txt
```

In this experiment, the proposed method is applied to an image file. The file was downloaded from website "The Honeynet Project" and written into the simulated NAND chip using the *nandwrite* command. Then we called *Tyaffs_ScanBackwards()* to perform the scanning in which the proposed method was executed.

This experiment demonstrates the effectiveness of our proposed method. However, we observed that not all the files were successfully recovered. We investigated the reason by manually checking the image using the hex editor and found out that some data pages of the unrecovered files were missing. One way of solving this problem would be to use pages that belong to other versions of the file according to the missing chunk IDs.

### 5.2 Evaluation on Efficiency

This experiment aims to show that the proposed method would consume less time compared to the reversed-scanning method [5] under the situation of fragmentation which is very common in the YAFFS2 file system. In the experiment, we first successfully acquired 8 image files from 8 Android smartphones that contain lots of user data using the DC-4500 forensic system developed by Meiya Pico Company [13]. We then reconstructed the 8 file systems by using our proposed method and the reversed-scanning method, respectively. The time consumed on completing the scanning of each image file using the respective method is shown in Figure 5 in which each blue bar represents the time consumed using our proposed method while the red bar represents that of the reversed-scanning method. Figure 6 shows the performance improvement of the proposed data recovery method over the reversed-scanning method in terms of time savings in the reconstruction of a file system. We could see from the figure that our proposed method could reduce the amount of time that is consumed on the reconstruction of a file system by about 20% on the average.

This experiment shows that the proposed method is more efficient in terms of the amount of time consumed on completing the scanning process. Moreover, we observed that the more fragmentation there is in an image file, the more efficient the proposed method can achieve.

## 6. Conclusion

In this paper, we proposed a file recovery method based on timestamps and Tnode tree in the YAFFS2 file system. According to object header and last access time stored in the flash memory, this method can locate data chunks physically based on the created Tnode tree. Experiments showed that our proposed method is both effective and efficient compared to some comparable method. In the future, we plan to address the issue of missing chunks and improve our method to further improve its effectiveness. We will also develop similar methods to apply to other file systems such as Ext4 used in the Android platform.

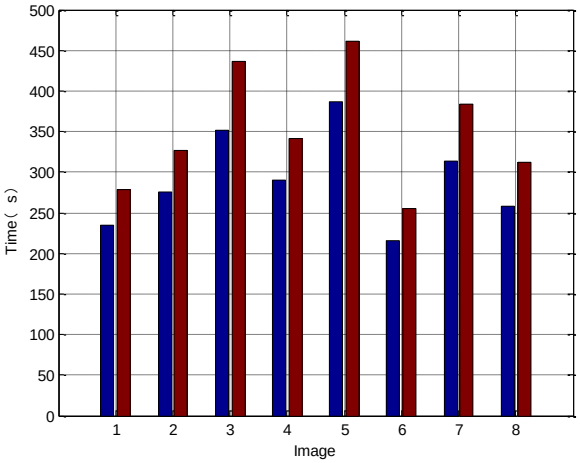



**Figure 5. Comparison of the Scanning Time**

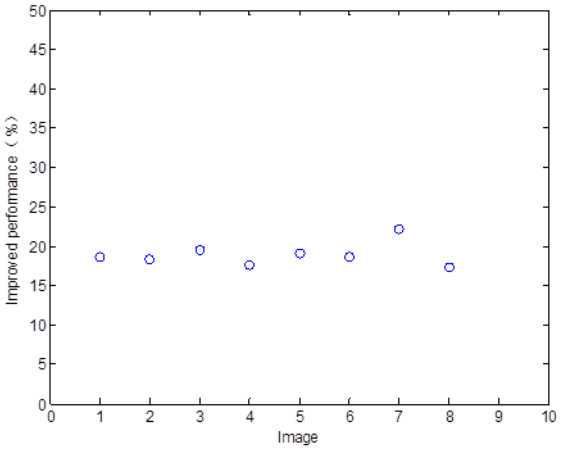



**Fig. 6 Percentage of Performance Improvement**

## Acknowledgements

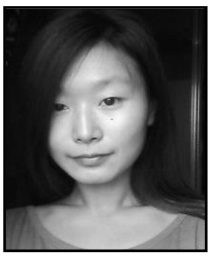
The work in this paper has been supported by National Natural Science Foundation of China (61272500), Beijing National Science Foundation (4142008), Shandong National Science Foundation (ZR2013FQ024) and Pre-launch of Beijing City Government Major Tasks and District Government Emergency Projects (Z131100005613030).

## References

[1] J. Luck and M. Stokes, "An Integrated Approach to Recovering Deleted Files from NAND Flash Data", Small Scale Digital Device Forensics Journal, vol. 2, no. 1, **(2008)**, pp. 1-13.
[2] C. Manning, "How YAFFS Works", **(2010)**, Available at: http://www.yaffs.net/documents/how-yaffs-works.
[3] B. Huang, "Data Recovery on Android Phones", "M. S. Thesis", The George Washington University, **(2011)**.
[4] L. S. de, F. C. Sicoli, L. P. de Melo, F. E. de Deus and R. T. de Sousa Junior. "Acquisition of Digital Evidence in Android Smartphones", Proceedings of the 9th Australian Digital Forensics Conference, Perth, **(2011)**, pp. 116-124.
[5] X. Yang, M. Xu and H. Zhang, "File Recovering from YAFFS2 based on Object Headers and Metadata", Proceedings of 4th International Conference on Graphic and Image Processing, Singapore, **(2013)**, pp. 876806 - 876806-8.
[6] C. Zimmermann, M. Spreitzenbarth, S. Schmitt and F. C. Freiling, "Forensic Analysis of YAFFS2", Lecture Notes in Informatics, vol. P-195, **(2012)**, pp. 59-70.
[7] S. Sack, K. Kröger and R. Creutzburg, "Overview of Potential Forensic Analysis of an Android Smartphone", Electronic Imaging, **(2012)**, pp. 8304M-8304M-11.
[8] J. Sylve, A. Case, L. Marziale and G. G. Richard, "Acquisition and Analysis of Volatile Memory from Android Devices", Digital Investigation, vol. 8, no. 3-4, **(2012)**, pp. 175-184.
[9] J. Park, H. Chung and S. Lee, "Forensic Analysis Techniques for Fragmented Flash Memory Pages in Smartphones", Digital Investigation, vol. 9, no. 2, **(2012)**, pp. 109-118.
[10] P. Dibb and M. Hammoudeh, "Forensic Data Recovery from Android OS Devices: An Open Source Toolkit", Proceedings of 2013 European Intelligence and Security Informatics Conference, Uppsala/Sweden, **(2013)**, pp**.** 226-226.
[11] B. Wu, M. Xu, H. Zhang, J. Xu, Y. Ren and N. Zheng, "A Recovery Approach for SQLite History Recorders from YAFFS2", Proceedings of Information & Communication Technology-EurAsia Conference 2013, Yogyakarta, Indonesia, **(2013)**, pp. 295-299.
[12] M. Xu, X. Yang, B. Wu, J. Yao, H. Zhang, J. Xu and N. Zheng, "A Metadata-based Method for Recovering Files and File Traces from YAFFS2", Digital Investigation, vol. 10, no. 1, **(2013)**, pp. 62–72.
[13] http://www.300188.cn/.

## Authors



**Na Huang**, is currently a M.S. student in the School of Software Engineering at Beijing University of Technology in China. Her research focuses on digital forensic.

**He Jingsha**, received his B.S. degree from Xi'an Jiaotong University in Xi'an, China and his M.S. and Ph.D. degrees from the University of Maryland at College Park in USA. He is currently a professor in the School of Software Engineering at Beijing University of Technology in Beijing, China and an associate director in the Low Carbon Research Center at Beijing Development Area Co., Ltd. in Beijing, China. Professor He has published over 200 research papers in scholarly journals and international conferences and has received over 40 patents in the United States and in China. His main research interests include information security, network measurement, and wireless ad hoc, mesh and sensor network security.