Forensic Science International: Digital Investigation 40 (2022) 301343

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2022 EU - Selected Papers of the Ninth Annual DFRWS Europe Conference

Quantifying data volatility for IoT forensics with examples from Contiki OS

Jens-Petter Sandvik [a, b, *], Katrin Franke [a], Habtamu Abie [c], André Årnes [d, a]

[a] Norwegian University of Science and Technology (NTNU), Norway
[b] National Criminal Investigation Service (Kripos), Norway
[c] Norwegian Computing Centre, Norway
[d] Telenor Group, Norway

ARTICLE INFO

Article history:

Keywords:
Digital forensics
IoT forensics
Contiki
Coffee file system
Evidence volatility
Triage

ABSTRACT

Forensic investigations are often conducted under limited resource availability such as time, equipment, and people. As data acquisition is resource-demanding already, a higher emphasis needs to be put on prioritizing the investigative steps to optimize the probability of collecting the relevant evidence. Data volatility measures how quickly data disappears from a system and is an essential part of assessing the likelihood of collecting the most valuable evidence. An investigator can use a model for the volatility to estimate the probability of the existence of evidence. This work motivates and details a model for data volatility and exemplifies it for the Coffee File System used in Contiki OS, an operating system for IoT devices. We conducted experiments to test how well the model corresponds to the collected simulated data and cross-validate the model with observations from file system operations. The results revealed that an approximated model based on the known workings of the file system underestimated the volatility. While there are many sources describing volatility qualitatively, there is little research on quantitative volatility, and this paper is a stepping stone to understanding a quantitative approach to evidence volatility.

1. Introduction

Internet of Things (IoT) concepts have existed for more than two decades, and it is now becoming more and more applicable in all areas of our lives. The technology can be found in many different systems, from smart agriculture, environmental monitoring, industrial systems, and smart infrastructures to smart home systems and personal devices. As IoT systems are more readily found everywhere, the probability of IoT systems being used for crimes or containing evidence from crimes is also increasing.

The devices' capabilities in an IoT system vary from vast clusters of servers running virtual machines and offering cloud solutions to small resource-constrained devices, running on low-capacity batteries, saving as much energy as possible. The dependency on network communication, the variance in the devices' capabilities, and the sheer number of devices increase the number of locations holding potential evidence. At the same time, the number of potential locations containing evidence increases, and the lack of standardized interfaces combined with protection mechanisms in devices, makes it more resource-demanding to collect evidence. Unfortunately, the resources available for most investigations are also constrained regarding time, equipment, and people.

To collect evidence from an IoT system that consists of both known and unfamiliar devices, the need for prioritizing the collection of evidence is prevalent. This prioritization is often called a *triage*. A triage in an evidence collection phase must assess the devices' likelihood to contain evidence, the accessibility of the data from the devices, and the evidence's volatility. From these three elements necessary for triage, we focus on the volatility of the devices' evidence in this paper and exemplify this with the Contiki Operating System (OS) and its Coffee File System.

A theoretical model for analyzing the volatility was applied and fed with numerical examples. The Cooja emulator was used to generate data on hardware emulated devices and to extract information about writing and erasure during the devices' operation. The write and erase operations were recorded to establish the lifetime of the data written, and these results were compared to the empirical results.

* Corresponding author. Norwegian University of Science and Technology (NTNU), Norway.
E-mail address: jens.p.sandvik@ntnu.no (J.-P. Sandvik).

https://doi.org/10.1016/j.fsidi.2022.301343

The objectives of this work are to answer the following research questions:

1. How can the data volatility in IoT devices be analytically calculated?
2. How can the data volatility in IoT devices be measured?
3. How well do the analytical results and measurements correspond?

This work focuses on the flash storage component of IoT devices, and in this study, the Contiki OS with the Coffee File System is used as an example case. Even though the size of flash memory is often tiny in IoT systems, the data volatility is usually lower[1] than in RAM. Other factors can affect the volatility but are not considered parts of this analysis, such as the physical destruction of the flash memory cells or controller; bad block management; bugs in hardware, software, or firmware; and bit flips caused by heavy ion irradiation.

The rest of the paper is organized as follows: *Related work* is background material of other research related to volatility. For a better understanding, we describe the *Coffee File System functionality*, and we continue to define *Data volatility*. Then we describe the *Experiments* and round off with a *Discussion*. In the *Conclusion* we summarize and suggest a way forward.

2. Related work

The concept of volatility is not new. The Internet Engineering Task Force (IETF) released a Request for Comments (RFC) on guidelines for evidence collection and archiving (Brezinski and Killalea, 2002). This document described the Order of Volatility (OoV) and how more volatile evidence should be collected before less volatile evidence to reduce the risk of overwriting the more volatile data locations.

The influencing factors for the volatility in a system can be many. Some of the factors are designed and documented; others are environmental events that can cause data to change or disappear. A study by Sandvik and Årnes (2018) showed and quantified how power outages in mobile phones affected registers holding the value of the real-time clock of the device.

Data in memory can change during acquisition while the system is running, which makes the resulting memory dump inconsistent, leading to a need for recording not only the spatial location of the pages but also the time of acquisition for each memory page (Pagani et al., 2019). Data inconsistency is one of the reasons for the data volatility, as memory pages might be overwritten during the acquisition process, making the memory structures inconsistent in the acquired data.

Even though the term *evidence volatility* has been used for a long time, it is not a well-studied field. However, in addition to the order of volatility, the term has been used to describe methods for acquiring evidence from volatile memory. Volatile data is often used to describe data stored in volatile memory, where the data disappears when the power is removed from the system (Sutherland et al., 2008). Data volatility is a description of the data's lifetime, as can be seen in the assessment of the *order of volatility*, found in many textbooks on digital forensics (Årnes et al., 2017).

The increasing complexity of volatile data in IoT systems was described as a challenge by Hegarty et al. (2014), where the author researches IoT systems that have several data locations with highly variable volatility for different locations. The transfer of data between devices in the network and the aggregation and processing of the data make it challenging to determine the data's origin.

The short lifetime of data in devices, together with the data being spread over a wide variety of locations in complex systems, are two forensic challenges that have not yet been fully solved. Montasari and Hill (2019) describe the forensic collection of volatile and processed user data from IoT devices as a future research direction.

The literature on evidence volatility is limited, and for the most part, it assesses the volatility on a qualitative basis. Apart from the qualitative assessment of the lifetime of data, there is little research on the quantitative assessment. Minnaard (2014) reported a notable exception to the above statement, where the author quantified the volatility of WiFi beacons sensed by the various wireless devices.

There has been done research on maximizing the volatility of data to protect privacy. Research into *secure deallocation* aims at increasing the data volatility in memory by nulling deallocated pages (Chow et al., 2005). A similar method can be used for nulling deallocated flash pages but is not considered in this paper.

The Contiki OS and the Cooja emulator/simulator was created by Dunkels et al. (2004) and have been the focus of several forensic research papers. Kumar et al. (2014) used Contiki devices for the network forensics research of Wireless Sensor Networks (WSN) running an IPv6 network with IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) encapsulation. However, as the focus was on routing information, the devices' flash memory was not considered.

Other authors have used the Contiki OS to make proof-of-concept implementation of forensic readiness systems. Hossain et al. (2018) implemented a system for automatically using a blockchain for evidence storage while the system is in regular operation. The nodes in the described system were running Contiki, and the nodes' data were collected and published to a blockchain by a Raspberry Pi in the same network. The focus was on the network and memory of the devices and did not consider the flash storage in the devices.

YAFFS2 is another flash file system studied from a forensic point of view. YAFFS2 is a file system designed for flash memories and does page reordering as a part of the garbage collection, and an in-depth view of the forensic artifacts has been reported (Quick and Alzaabi, 2011; Zimmermann et al., 2012). In addition, a method has been proposed for recovering the version history of files in the YAFFS2 file system, using the embedded system information in the file headers (Li et al., 2016).

3. Coffee File System functionality

The Contiki OS is an open-source operating system designed for resource-constrained devices, such as sensors in a WSN. It has an integrated 6LoWPAN network stack for communication where the physical network layer and the link layer do not support data frames big enough to accommodate IPv6 packets without compression and fragmentation. The OS footprint is about 100 kB, and it needs at least 10 kB of RAM to run.[2] The codebase is approximately 112 000 lines of C code, including headers.[3] Contiki-NG is the descendant of the Contiki operating system, and in this paper, the name Contiki is used for Contiki-NG unless otherwise specified. The forensic artifacts of the Coffee file system have been described in detail by Sandvik et al. (2021a), but we will give a short description of the most relevant findings here.

Contiki includes a flash-based file system called the Coffee File System, designed to minimize the resources used and spread the

[1] A lower data volatility means longer life of data.

[2] https://github.com/contiki-ng/contiki-ng/wiki, visited 2020−08−25.

[3] Counted using the program "cloc" on the os/directory of the source tree.

writing not to wear out individual flash pages. The wear leveling is managed by letting all writes happen at the end of the file system, marking obsolete pages ready for Garbage Collection (GC). Wear leveling is important in Flash memory, as each cell only can be programmed a given number of times before it starts failing, and it will spread the writing such that all cells have about the same number of writes over the lifetime of the file system. A page is the smallest addressable unit for writing and is dependent on the memory chip. A memory cell can be programmed by flipping the value it holds from one to zero, but a whole sector has to be erased at once to reset the value. A sector contains several pages,[4] and is the smallest addressable unit for a reset or erase.

Writing files will initially allocate 17 pages for the file, and if the file is modified, an additional log file of five pages is allocated to accommodate four modifications. When the file is modified four times, a new file and logfile compound is allocated, and the content of the last version is copied to a new base file. The old version is then marked as obsolete, and when the writing reaches the end of the file system, the GC is called to erase all sectors containing only obsolete pages. The file system does not copy active pages to free up more sectors, which means that if there is one active page in each sector, the file system cannot erase any sectors.

Contiki comes with a simulator called Cooja. This simulator can compile the firmware for the native architecture of the computer running Cooja and run simulated devices very fast and efficiently. Another option is to compile the firmware for the target architecture and run the code in an emulator within Cooja. The advantage of the former approach is efficiency, while the latter's advantage is the resemblance to the physical implementation. In our work, a hardware architecture in Cooja with an accessible flash memory that can be exported to a file is used.

4. Data volatility

Data stored in a system will eventually disappear, and the time from the data appears until it disappears is the lifetime of the specific data. The probability distribution of the lifetime of a class of data can therefore express the volatility of the data. The expected mean of this probability distribution is a measure that gives us some information about the lifetime of the data and can be used as a measure of the data's volatility (Sandvik et al., 2021b). As the expected value does not convey information about the *spread* or shape of the distribution, the measure is not perfect but gives enough information for an investigator to get an understanding of the likelihood of finding existing evidence.

This study is based on the volatility model introduced by Sandvik et al. (2021b). The authors did not go into specific file system details, so this study extends the defined model by showing the application of the theoretic model. This study focuses on the non-volatile parts of the memory in devices, but it can also be extended to cover volatile memory and distributed systems. Other storage media such as Random Access Memory (RAM) or cloud storage systems all share the same functionality: loading data into the memory, processing the data, moving data, and keeping track of the active storage locations. The same general model can be used for these types of storage, but each type of storage system needs to be assessed and analyzed individually.

By knowing or estimating the volatility in the devices and data locations, the investigator can better understand where evidence might be found and prioritize the acquisition and examination phase of the forensic process. One method for using the volatility can be to list all known devices in the system under investigation, order them by the volatility and probability for the device to have contained evidence, given the investigation hypothesis. A high probability for the evidence to have been on the device combined with high volatility means that the device should be acquired before any potential data is lost.

The volatility depends on the method used for collecting evidence: the abstraction level it uses and the amount and type of data it changes during its operation. A logical data collection sees the active data and will not extract any deleted data. On the other hand, a physical collection can see both the active and the deleted data if the data has not been erased. We need to know which collection method the volatility describes to express the volatility adequately. In this study, we assume a low-level collection method that makes a bit-by-bit copy of the contents of the flash memory, either a physical acquisition method or a pseudo-physical method (Klaver, 2010).

4.1. Data lifetime

The unit for measuring the data's lifetime can be selected from at least three types of units: The *number of clock cycles* can give us the time independent of the CPU's speed, with the drawback that the wall clock time needs to be computed for the average number of clock cycles per time unit. Another measure that can be used is the *number of operations* affecting the data, which is easier to analyze, but the drawback with this approach is that the investigator still has to assess the number of file operations per time unit. The third measure, which we use for this paper, is the *wall-clock time* passed before the data is unavailable. This third option is the available measure for an investigator, as all assessments about triage or available time depend on this.

The data's lifetime depends on two phases: the period the data is alive in the system, and the period the data exists in the memory before it is overwritten. The former phase can be considered when the data is being held by *purpose*, the latter is when the data is being held by *chance*, where purpose signifies that the application keeps the data active for a purpose, and chance refers to that the data still exists in a location that is not yet reclaimed by the GC because at least one other page in the same sector is still active. A basic equation for the lifetime of the data in a specific medium can be formulated as:

$$T_{\text{Data}} = T_{\text{Delete}} + T_{\text{Erase}} \tag{1}$$

where T is a probability distribution over a period of time, and the relation between these term are shown in Fig. 1.

The first part of Equation (1), T_{Delete}, is the time from the creation of the file to the deletion of the file or the modification of the file contents. The time before deletion often has a high variance between applications and depends on how the running application modifies the file. As the timescale in question is from microseconds to years, it is impossible to give a generic model for the expected time without knowing the running program's behavior. A configuration file will typically have a high expected time before being modified or deleted. The expected lifetime for data in a sensor

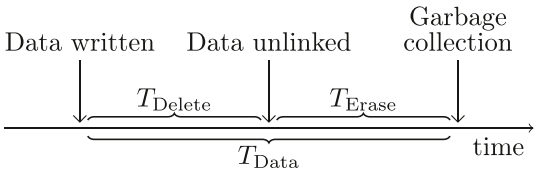


Fig. 1. Expected lifetime of data as a sum of the expected lifetime of the file and the expected lifetime of deleted data.

[4] default for Coffee File System is 0x100 (256) pages per sector.

depends on the sampling rate, the amount of memory, and how often the data is sent to a central server, and the data can exist on the device from a few seconds to several hours. As long as the content remains in an active file, there will be at least one instance of the data in the file system.

The second part of Equation (1), T_{Erase}, is the lifetime of inactive data in the file system. This data is available until the sector[5] is erased, which typically happens when the GC erases used sectors so that they can be reused. The time from a page is deleted until the garbage collection erases the data in that page is isolated and studied in this research.

Some file systems will aggregate and copy active data from sectors containing mostly deleted pages to new sectors. By combining live pages into one sector, the other sectors can be erased by a garbage collection and reused. Other file storage functions can include remote storage of data or data encryption. As described in the model by Sandvik et al., the storage management functions can copy, erase, encode, or encrypt data as a part of the operation of the file system or flash management (Sandvik et al., 2021b). The term *delete* in Equation (1) can therefore be thought of as rather the term *unlinked*, as data can continuously be copied and the old copy unlinked from the file system.

As an example, data can be moved within the system, which creates two locations of the data: one that is active in its new location and the original data location that is unlinked and will be erased after some time, when it is overwritten or the GC frees up the sector containing the data. The live data copy is still accessible for the file system and will not be erased before the file is deleted, the hardware fails, or the data is moved to a new location.

The added complexities of management functionalities like this depend on the method used for collecting the data. The number of deleted copies of some data is irrelevant for a logical file copy from a running system, as the collection method cannot access this information.

This simple model does not take into account the probability of physical failures. As these, in general, are relatively low probability events, they have a very small, albeit non-zero, impact on the lifetime of the data.

4.2. Volatility model

The knowledge about the volatility of the Coffee file system can be implemented in the volatility model as described by Sandvik et al. (2021a). The model considers differences between how memory devices handle data and the acquisition method used to collect the evidence. In the model, an application uses the data with its probability distribution of the data volatility. The storage system will have its view of the data, both in terms of internal copying as part of the regular operation of the file system and the time it takes before the file system will overwrite the data. The physical devices and the failure rate are at the bottom of the storage stack, which describes the probability of the data disappearing due to physical failures.

The model by Sandvik et al. (2021b) defines the volatility model as a six-tuple: $\mathcal{V}\_\{\mathcal{D}\} = (L,E,A,M,D,S)$. L is the storage abstraction layers, E is the events that trigger data to be stored, A is the application's write and deletion functions, M is the set of functions that manage the storage, copying and moving data in volatile or non-volatile memory, D is the device's physical reliability, and S is the environmental aspects affecting the volatility of the data.

The abstraction layers in L can be described as the three writing applications at the top layer, followed by the Coffee File System and the physical flash memory. In this paper, we do not consider the contents of the RAM, but for completeness of the discussion, the abstraction layers include the memory management functions in the Contiki operating system.

In this work, the application activity function, A, is three processes writing, each with a randomly selected, constant writing rate between 1 s/w and 1200 s/w. The events, E, are just the internal clock triggering the writing process in our experiments.

The set of storage management functions, M, write the modified data to new locations every writing operation, and for every fourth operation, the current content of the whole file is copied to a new location. When the writing reaches the end of the file system, the garbage collection is invoked, erasing all sectors that do not contain any active pages.

In this paper, we do not experimentally test the device reliability, as emulated devices effectively isolate the system from physical failures. The simulated system also removes most of the environmental effects from the system, as the physical and operational environment do not affect the data writing or storage, and only the system's configuration has an impact. The impact from the configuration is due mainly to differences between the emulated devices and the physical ones, such as the speed of the memory operations due to other physical properties.

As long as the application keeps the data alive, it should be available in the flash memory unless a physical failure or bug prevents it from retrieving the data. The data can, during its lifetime, be copied to new locations in the flash memory, which increases the probability for at least one copy to survive for a longer time. In the worst case, the probability is the same if all copies are erased simultaneously, and the failure rate for individual memory addresses is negligible.

To simplify the initial analysis, we assume that the data only exist for one writing operation and is modified in the next operation, which means that the application only keeps data active for a period similar to the writing rate of the application.

4.3. Volatility in the Coffee File System

The Coffee file system used in Contiki OS is designed for flash memory in resource-constrained IoT devices; it balances simplicity in design and run-time complexity with usability. The advantage of using a relatively simple file system in the initial analysis is that all elements that affect the volatility can be controlled.

The time between two consecutive garbage collections is dependent on two factors. One factor is the combined writing rate of the files and the number of records in the log files, as the garbage collection will start when the writing reaches the end of the file system. This can be expressed as:

$$r = \frac{1}{\sum_{i=1}^{F_n} \frac{1}{r_i}} \tag{2}$$

$$T_{\text{GC, first}} = r\left(\lfloor \frac{S_S S_n}{F_S} \rfloor L_r + 1\right) \tag{3}$$

where r is the combined write rate, given as time units per write, F_n is the number of files in the file system, r_i is each files write rate, T_{GC} is the time between garbage collection runs, S_S and S_n is the size of a sector and the number of sectors in the file system, respectively, F_S is the average file size, and L_r is the number of log records written before a new file is created, obsoleting the previous base and log file pair.

The other factor is how many sectors that are available for allocation. Equation (3) uses the total number of sectors, but this

[5] Often called an erase block in flash memory terms.

gives the time from writing starts in an empty file system until it reaches the end. Unfortunately, this is only a valid assumption between the first time writing starts in the file system until the first garbage collection. After this, there is at least one sector containing active pages in the file system.

The minimum and the maximum number of sectors containing active pages in the file system are dependent on the number of files in the file system and the file sizes. For one file, the number of sectors containing active sectors is given by the file size. If it is less than half of the sector size, only one sector contains active pages when the GC starts, and if it is bigger than half of the sector size, the number of sectors with active pages is given by the file size.

More generally, the minimum and the maximum number of sectors containing active pages when the garbage collection runs is given by the tightest packing of files in the last sector for the minimum amount of sectors surviving garbage collection. For the maximum number of surviving pages, each file has to extend into another sector with at least one page, and each sector can only contain the pages belonging to one active file. The above description can be described mathematically as:

$$N_{\min} = \lfloor \frac{\sum_{i=1}^{F_n} F_{S,i}}{S_S} \rfloor \tag{4}$$

$$N_{\max} = F_n + \sum_{i=1}^{F_n} \lceil \frac{F_{S,i} - 1}{S_S} \rceil \tag{5}$$

where $F_{S,i}$ is the size of the ith file.

Unfortunately, the average number of remaining pages is more difficult to calculate, as it depends on the number and location of remaining pages from the previous garbage collection. Therefore, we use the outer limits given by the maximum and minimum described in equations (4) and (5) here.

The time between two consecutive GC runs is dependent on the number of remaining sectors, and given the limits to the number of remaining sectors in Equations (4) and (5), the minimum and maximum time between GC can be calculated as:

$$T_{\text{GC, min}} = r\left(\lfloor \frac{S_S(S_n - N_{\max})}{F_S} \rfloor L_r + 1 \right) \tag{6}$$

$$T_{\text{GC, max}} = r\left(\lfloor \frac{S_S(S_n - N_{\min})}{F_S} \rfloor L_r + 1 \right) \tag{7}$$

As files are written to the file system, the newest version is always written at the end. If the modified file is in a sector where it was the only active file, that sector will be erased during the following garbage collection. The longest time a file will survive in the file system is therefore given by the file with the lowest write rate, where the write rate is given by the time between write operations.

The survival time of the pages in a sector is given by the time between garbage collections multiplied by the number of garbage collections the sector will survive. The number of garbage collections a sector survives is dependent on the file with the slowest write rate. The maximum survival time can be summarized in the equation:

$$T_{\text{sect, max}} = T_{\text{GC}} \lceil \frac{r_{\max}}{T_{\text{GC}}} \rceil \tag{8}$$

where $T_{\text{sect, max}}$ is the time a sector would survive, T_{GC} is the time between GC, and $r_{\max}$ is the longest writing rate.

The lifetime of unlinked data in the file system is therefore correlated with the writing rates of the individual files in the file system. The lifetime of the data is dependent on the time between the garbage collection runs. A fast total write rate will also reach the end of the file system faster and trigger the GC earlier. Given the time between GC and the write rate of the individual files, the average time a sector containing a file is retained is given by the write rate of the file in the sector divided by the time between garbage collections. The number of garbage collections each file contributes is thus given by $\frac{r_f}{T_{\text{GC}}}$.

A numerical example is where three files are written with a rate of 1 s/w, 50 s/w, and 1000 s/w, respectively, which will give a combined writing rate of $r \approx 0.979\ s/w$. Given a sector size of $S_S = 256$ pages, $S_n = 15$ sectors in the file system, a file size for all files of $F_S = 22$ pages, and $L_r = 4$ records in the log file, the time between GC is given by equations (3), (6) and (7) to be approximately 682 s for the first GC ($T_{\text{GC, first}}$), and between 635 s and 408 s for the other GC intervals ($T_{\text{GC, min}}$ and $T_{\text{GC, max}}$, respectively). With a write rate of 1000 s/w, this means that there is a high probability for at least two sectors to survive a GC: the sector containing the file with the highest write rate and the sector containing the file with the lowest write rate. There is also a high probability for a sector surviving two garbage collections, as 1000 s/w gives a number of survived garbage collections per write between $\frac{1000\,s/w}{635\,s/\text{GC}} = 1.57\,\text{GC}/w$ and $\frac{1000\,s/w}{408\,s/\text{GC}} = 2.45\,\text{GC}/w$.

4.4. Approximation of volatility

The analysis of the lifetime of unlinked data is non-linear because of the floor and ceiling functions and can become quite complex, even for relatively simple file systems such as the Coffee File System.

The volatility of a single device is dependent on how fast the writing reaches the end of the file system and the number of garbage collection cycles some pages survive. For the Coffee File System, a simplified model can summarize a uniform distribution of the lifetime of the deleted pages after one garbage collection routine, the average number of pages surviving one garbage collection, and the number of pages surviving the calculated max number of garbage collections.

The sector containing the writing pointer in the file system is retained when a garbage collection is started, and this retained sector does not impact the volatility. The retention of this last sector does not impact the volatility, as the sector retained means one fewer sector for the next inter-GC time. This is also true for more than one sector, as long as the retained sectors are consecutively written sectors: the two last sectors written will lead to two fewer sectors that are writeable in the next period, but the difference in time between the last write and first write, $S_{(S_n-2)} - S_{-2}$, is the same as the difference in time between the last sector and first sector in case of no retained sectors, $S_{(S_n)} - S_0$.

If, on the other hand, another non-consecutive sector is retained, the lifetime of the data will be affected as sectors older than the time between garbage collections will be retained. This

happens regularly in the Coffee File System, as the garbage collection process seems to skip a sector if it is the only sector that can be retained. Since the last written sector is retained, the first GC retains the last sector, but during the next GC both the last written sector (the next to last sector in the file system) and the last sector in the file system (retained from last GC) are both retained. This behavior has been observed in all tests so far.

The writing behavior would give an approximately uniform distribution with a small burst of lifetimes around the lifetime of the sectors surviving several garbage collections. The approximated distribution of page lifetimes can thus be described as:

$$T = \frac{rk}{2}\left(\lfloor\frac{S_n S_S}{F_n}\rfloor L_r + 1\right) \tag{9}$$

where k is a scaling factor to account for the non-consecutively written sectors that are retained. The last part of the equation is the average number of retained divided by 2, as this is a uniform distribution and the average for an uniform distribution is given by (max-min)/2.

5. Experiments

The experiments were performed using the Cooja simulator with emulated devices. The functions `erase_sector` and `program_page` from the file `arch/platform/sky/dev/xmem.c` in the Contiki sources were patched to log writes and erasures of canary tokens. As canary tokens, 32-bit values were used for detecting the file writes and erasures, and each file modification wrote a canary token to the first page of the file. The patched functions scanned the pages written and erased for the canary token values. When the canary token was found during a file system operation, the page offset was logged to the simulator output, and a Python script was designed to analyze the results.[6] Each writing operation contained the canary, and each file had a unique canary token, which the patched file system reported.

5.1. Setup

The experiment tested the lifetime of data after the application had written to the file, unlinking the old version of the data. The experiment was done by modifying three files in each device, where each file had a different modification rate. The log from the simulator contained all write and erase events for the canary tokens associated with the files, and a Python program for analyzing this file was written. It should be noted that the canary token samples the lifetime of pages in the system, and thus, when using the term *lifetime of pages*, it is the same as *lifetime of the canary tokens* throughout this paper.

Each simulation consisted of 16 devices, each with three files being written. The write rate of the files is hard-coded at compile-time, and each file had a write rate chosen at random between 1 and 1200 s per write, picked from a uniform distribution. The simulation was done with a new set of devices ten times, and each simulation was run for 864.000 s (simulation time) which equals ten days. In total, the simulated devices were run for a total of 10 days $\times$ 10 simulations $\times$ 16 devices, or 1600 days.

One of the devices failed/crashed during the experiments and was subsequently removed from the rest of the analyses. The writing rates of the files in the removed device were 273, 646, and 1046 s/write, and the error encountered was an interrupt from the watchdog timer after 78 h of simulation time.

5.2. Observed write rates

Each device's actual combined writing rates were recorded and compared with the programmed writing rates. The combined writing rate is the total writing rate of all processes' write operations in a device combined and seen as one process. The observed writing rate was found by dividing the simulation time by the count of all write operations logged for a device during the simulation.

Fig. 2 contains a plot showing the relation between the programmed writing rate for each device and the observed writing rate. Table 1 shows the average ratio $\frac{r_{obs}}{r_{prog}}$ together with the max/min values and quartiles. The expected result was that the observed result should be approximately close to the programmed rate.

The observations are at approximately 80% of the programmed write rate. This difference is stable for almost all devices regardless of writing rate, with one exception where the programmed writing rate was 4.741 418 writes second, and the observed rate was 2.509 141 writes/second, a ratio of 0.529 196. This device had the fastest writing rate among the simulated devices.

The reason for this discrepancy in programmed and observed writing rates is unknown. The log data from the experiments revealed a burst of writes after garbage collection, possibly to make up for the lost writes during the garbage collection. If the burst lasts longer than necessary, it might help explain at least a part of the discrepancy between the programmed and observed writing rate. Another possibility for the discrepancy is that the emulated real-time clock runs faster than the simulation time, thus writing consistently with a higher rate.

5.3. Garbage collection

The time between each GC can be named the GC-to-GC time, or the inter-GC time. The inter-GC times are calculated in Equations (6) and (7). This analysis of the simulations will show how well they correspond with the observed inter-GC times.

The GC also uses time on running, and this time is in this work referred to as the GC running time or GC time. The median GC time was 56.8 s for all motes in all simulations, with a minimum of 13 s

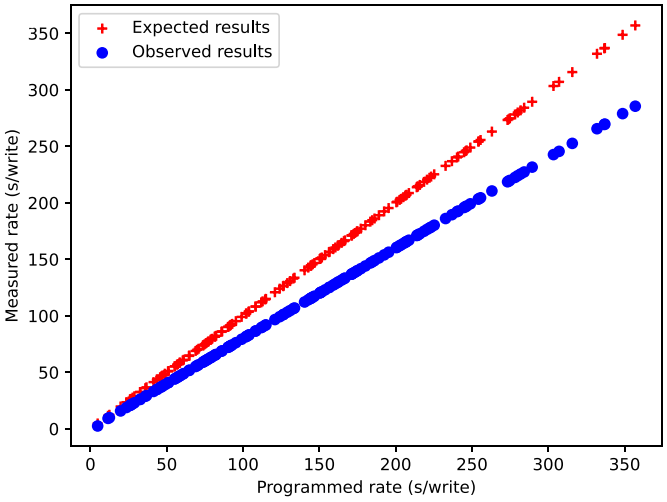


Fig. 2. The programmed writing rates for each device plotted together with the observed writing rate. The plot marked with "+" shows the ideal rate if the observed and programmed rate is identical.

[6] The script is published at https://github.com/jenspets/contiki-ng_motes, under the *volatility* folder under a GPLv3 license, and the dataset is shared under a CC-BY-SA license.

Table 1
The quartiles and average of the ratio between the observed and programmed volatility.

Average	0.798 277
Std.dev	0.021 475
Min	0.529 196
25%	0.799 935
Median	0.799 986
75%	0.800 021
Max	0.800 331

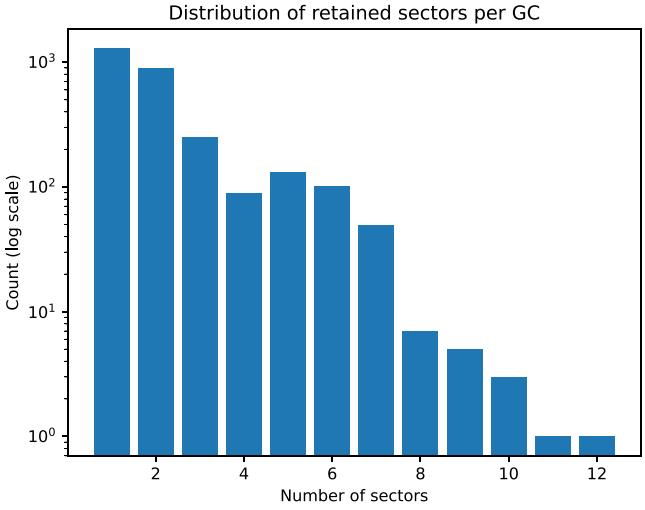


Fig. 3. The histogram of retained sectors after each garbage collection in all devices.

and a maximum of 1 min and 13 s. The quartiles were 56.7 and 61.1 s, which indicates that the GC time was stable.

The theoretical number or retained sector is found in Equations (4) and (5). Fig. 3 shows the histogram of the number of retained sectors after garbage collection for all devices in all simulations. The average was 2.11 sectors retained per GC, the minimum and the maximum number of retained sectors were 1 and 12, respectively, and the upper and lower quartiles were 1 and 2 retained sectors, respectively. The highest observed number of retained sectors, 12, was higher than the calculated maximum, 6. There is only one mote out of the 160 that retains more than five sectors during a GC, and the write rates of the files in this device were 5 s/w, 104 s/w, and 774 s/w. This might indicate either that the garbage collection started before the file system was filled or that the garbage collection was interrupted before it finished.

A closer examination of the GC run that left 12 sectors showed that it only used 13 s on the garbage collection and that it collected the first two and the last sector. The reason for the short garbage collection run is not apparent from the logged data or the source code of the operating system. The sectors collected during this run were sectors number 1, 2, and 15. As the garbage collection source code shows that it iterates over all sectors, erasing them if there are no active pages in the sector, there seems to be a function in the file system code that mislabels some sectors under certain conditions, skipping them during garbage collection. There does not seem to be any processes that cancel the GC, as a cancellation would not proceed to erase the last sector in the system, so misclassification of deleted pages by the garbage collection system initially seems plausible. The sectors are eventually erased during later garbage collection runs, which does not support the misclassification hypothesis, as something needs to be changed for the classification to change.

The inter-GC time for this mote was calculated to be between 1484.1 s for six retained sectors and 2309.1 s for one retained sector, and the experimental results for the mote showed an inter-GC time of between 1263.4 s and 3306.0 s. The more retained sectors from one GC, the fewer free pages there will be in the file system afterward, which explains the shorter time between garbage collections. The longer retention time is still unexplained but might be caused by the operating system using longer time on finding allocatable areas due to fragmentation or retaining sectors for more GC cycles than expected.

The values $T_{GC,\ min}$ and $T_{GC,\ max}$ in Equations (6) and (7), respectively, are the shortest and longest time between garbage collection runs calculated from the write rate. Table 2 shows the quartiles for the shortest observed inter-GC time minus T_{min}, and T_{max} minus the observed maximum time for all 159 nodes together with the theoretical minimum and maximum time, given the observed write rates. The differences between the model and the programmed/observed extreme values are denoted by ΔT and ΔT_{obs}, for the programmed and observed values, respectively. The terms *Min* and *Max* are used for the calculated model values, where $\Delta T_{min} = \text{Min} - T_{min}$ and $\Delta T_{max} = T_{max} - \text{Max}$. A negative value thus indicates an observed value outside the theoretical limits from equations (6) and (7).

The observed minimum time is higher than the calculated, with a median of 3.5 h longer than the minimum calculated. The calculated maximum time is also shorter than the observed time, with a median of 2 h, and here all nodes were longer than the theoretical maximum, with the longest observed time 4.5 h longer than the theoretical maximum. This means that the model developed for the time between GC runs are predicting too short values, both for minimum and maximum values.

It is interesting to note that using the observed writing rates rather than the programmed writing rates predicts the maximum retention time even worse, with a median error of approximately 7 h shorter than the actual maximum retention time (25 123 s). The minimum time between garbage collections was much higher than the predicted time, except one about 20 s shorter than the predicted minimum.

Using the programmed rate, the time to the first GC is close to the observed time. The Mean Squared Error (MSE) between the calculated time to the first GC and the observed time to the first GC was 182 162.5. Using the observed writing rate, the MSE was 538 053 648.0, three orders of magnitude greater than the result using the programmed writing rate. Table 3 shows the error when calculating the predicted time to the first garbage collection minus the observed time to the first garbage collection for each mote.

5.4. Page lifetimes

The average retention time for the pages, or more precisely the average lifetime of the canary tokens in all the simulated devices, was 7 h, 32 min, and 27 s and the maximum retention time of all the simulated devices was 5 days, 8 h, 58 min, and 41 s. Fig. 4 is the

Table 2
The difference between the observed min/max times between garbage collections and the calculated times.

Percentile	ΔT_{min}	ΔT_{max}	$\Delta T_{obs,\ min}$	$\Delta T_{obs,\ max}$
Min	−1168.9	−16 054.2	−21.4	−61 687.9
25%	6576.7	−9608.7	14 283.3	−36 401.7
50%	12 692.5	−6379.5	27 347.0	−25 123.1
75%	18 836.0	−3423.5	39 934.4	−13 252.6
Max	32 415.4	−228.8	69 082.8	−1677.5

Table 3
Statistics of the difference between the observed time to first GC and the calculated time for all simulated devices. MSE is the Mean Squared Error.

Statistics	Programmed rate	Observed rate
MSE	182 162.5	538 053 648.0
Median (s)	171.533	−19 580.139
Min (s)	−381.987	−47 983.583
25% (s)	46.137	−28 480.915
75% (s)	373.093	−10 357.329
Max (s)	1835.551	−1557.100

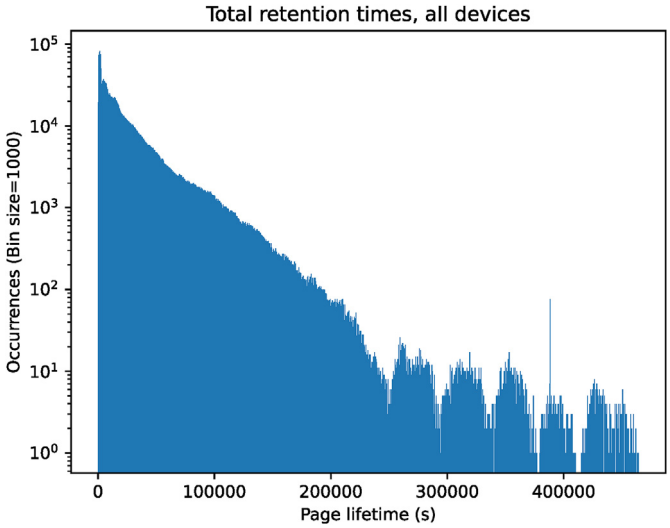


Fig. 4. Histogram of page lifetimes in all devices during the simulations. Please note the logarithmic scale.

Table 4
The difference between calculated and observed average lifetime of pages in the devices.

Statistics	P	O	PR	OR
k	1.065	1.331	1	1
Mean	−51.150	−66.700	−3351.295	−13 509.640
Std.dev.	406.690	409.645	1920.197	7771.355
Min	−1627.143	−1658.172	−9254.061	−33 780.764
25%	−217.455	−229.048	−4503.557	−18 900.933
Median	−3.147	−6.504	−3096.179	−13 158.907
75%	135.829	112.541	−1834.612	−6964.147
Max	961.772	951.798	−254.447	−1032.396

histogram of the canary lifetimes for all devices for all simulations, and we can see a distribution of lifetimes that follows a distribution resembling a negative exponential distribution with a fat tail.

The expected lifetime of the pages can be calculated using Eq. (9). Table 4 shows the difference between the calculated and observed average lifetime of pages in the devices. The columns show the differences between the calculated and observed lifetime of the pages, using the programmed (P) and observed (O) writing rates, together with the results where the scaling factor, k, is 1. An average of 0 in the table means that the calculated and observed rate is the same. As the programmed and observed writing rates differed, the scaling factor, k, in Eq. (9) was found for both rates by minimizing the median differences between the observed and calculated values and rounding the value to three decimals.

The results show a variance in the difference between the calculated and the observed lifetime. The variability is similar between the calculated lifetimes, and even though half of the devices were within 6 min of the calculated lifetime, the standard deviation of over 6 min shows that some lifetimes are closer to the extreme values. Still, a worst-case misprediction of less than half an hour is where the experimentally observed lifetime of the data was 126 591 s, which gives a misprediction rate of approximately 1.3% for this device.

5.5. Distribution of lifetimes

The average and quartiles of the pages' lifetime indicate the spread of the data, but we also should test whether our assumption of a uniform distribution holds. Eq. (9) assumes a uniform distribution when calculating the average, and a deviation from this can affect the result.

Even though the distribution of total page lifetimes in all devices has a negative exponential tendency, a Kolmogorov–Smirnov test of the combined lifetime did not show that the lifetimes were from a negative exponential distribution. A p-value of close to zero shows that the hypothesis that the distribution is a negative exponential distribution is highly improbable.

The lifetimes of each simulated device were then tested. All but three continuous distributions in the SciPy library were fitted to the closest matching parameters.[7] A Kolmogorov–Smirnov test is a statistical test for continuous distributions that tests whether the observed data is drawn from a particular distribution. The Kolmogorov–Smirnov test was then used on each set of distributions and its fitted parameters. All p-values were zero, meaning the probability almost certainly was not drawn from any of the distributions in the SciPy package.

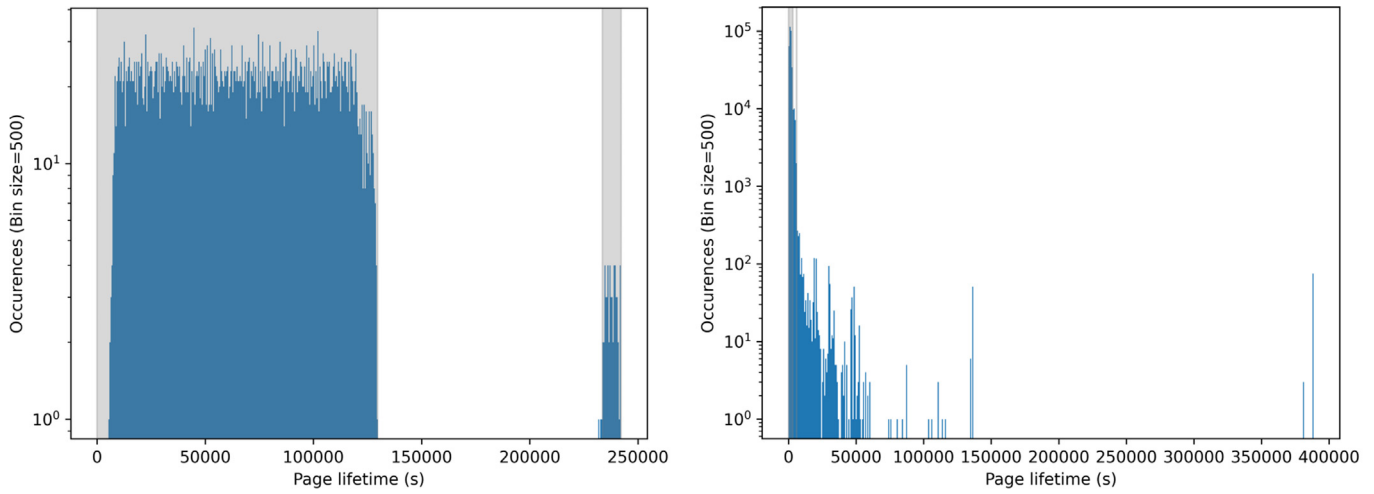
The lifetime of the pages carrying the canary token depends on non-linear functions, where the majority of pages follow a uniform distribution over the inter-GC time. Thus, the probability for the observed values for each device to be drawn from any of the continuous distributions from the SciPy package was low, and the hypothesis that the observations were drawn from any of these distributions was not supported.

Fig. 5 shows the histogram from two of the devices, with marked areas in gray signifying the expected range of lifetimes. A typical result is shown in Fig. 5a and 5b shows the device with the most page lifetimes outside the marked areas, where 8.2% of the observations were outside. The first gray area is from zero to the calculated time to the first GC, using the programmed writing rate. The second gray area is thinner and is given by the time to the first GC divided by the number of sectors and transposed to the slot given by the number of sectors minus three. The third to last sector is based on the observation that the measured average number of retained sectors after a GC is 2.97.

Table 5 shows the statistics of the number of measured page lifetimes inside and outside the previously defined range. For each device in each simulation run, the number of observed lifetimes matching the expected ranges and the number of observations outside the expected range were collected. The percentage of observations outside the expected range was counted for each device, and the statistics were calculated.

The results show that none of the built-in distributions in SciPy could describe the data with a significant p-value, and the reason for this is probably the observed near-uniformity in most lifetimes but with an extra burst of lifetimes close to two times the inter-GC times. As the burst contain relatively few lifetimes, the average should be close to the one predicted by a uniform distribution.

[7] Some distribution fitting timed out after 5 h and were excluded. These distributions were: kstwo, levy_stable, and studentized_range



(a) A typical node.

(b) The least fitting node. The two gray areas are barely visible at the peak to the left side.

Fig. 5. The histogram of page lifetimes from two nodes.

6. Discussion

Volatility is a term that is often used but seldom quantified. This paper's definition and model are only valid for the Coffee file system and need to be adapted for other file systems and more complex file operations. As nodes are resource-constrained and file system operations are costly in terms of the available resources, the amount of data written to a file system is generally low. However, with the introduction of intermittent computing, data needed for operation needs to be stored in non-volatile memory. More complex file operations and a more generic model should be developed for future work.

For the volatility model to be helpful for an investigator or first responder during the busy start of an investigation, the model described here has to be developed further and simplified such that the investigator does not need specific domain knowledge about all OSs, memory layouts, and file systems in order to quickly assess the volatility in the various parts of system. A model that is hard to use and needs a high degree of specialized expert knowledge will seldom be used, but a model that is easy to understand and needs little resources to use will give an investigator a better understanding of the probabilities of finding evidence in the case.

There is a discrepancy between the programmed and the measured writing rate, but this did not show up in the distribution analysis, where the programmed rate fit the data better than the observed rate. This was a surprising result, and the observed writing rate was calculated with a Python script and verified with an AWK script to count the number of writes during the ten-day period the nodes were running. This is a finding that should be studied further.

The focus of this research was how a volatility model could be used for analyzing the Coffee File System. The model can be more beneficial for practitioners to assess systems for probable data locations and prioritize collecting and analyzing evidence by analyzing more systems to find their lifetime distributions.

Table 5
The aggregated statistics over percentage of observations outside the expected range from each device.

Count	159
Mean	0.432 8
Std.dev.	1.120 7
Min	0.041 5
25%	0.111 5
Median	0.145 1
75%	0.183 0
Max	8.201 4

7. Conclusion

We examined the volatility of files in the Coffee File System and created a model of the volatility that can be used to analyze the lifetime of the data between deletion and erasure of the file contents. The volatility can thus be analytically calculated by either calculating the volatility based on the file system implementation or by approximating it using simplifications to the system model. The findings show that even a relatively simple file system and deterministic processes writing to the file system generate non-linear and complex distributions of data lifetimes. These questions cover the first research question.

The lifetime of the data used to measure the volatility was collected by patching the memory writing and erasing functions in the Contiki operating system. The patched functions logged the writes and reads of predefined canary tokens to a log file with a timestamp and the location in the flash disk. This file was further analyzed to find the lifetimes of each page in the file system, addressing the second research question.

We have also shown how the data lifetime can be approximated using knowledge about the file system and the data writing processes. The analytically found lifetimes were compared with the measured lifetimes, but the upper bounds of the lifetimes were, for the most part, underestimated, as the measured maximum lifetime was much higher than the analytically calculated. To conclude the third research question, the observed lifetimes were greater than the expected lifetime, and there were some huge outliers that could not be explained by the analytical model.

This study represents one stepping stone to understand the concept of volatility in systems and for the investigator to estimate the probability of finding evidence in a complex system. Future work needs to set this into a bigger perspective and consider even more evidence locations and data dependencies in the system.

Acknowledgement

Thanks to Arvind Sharma at NTNU for valuable feedback on the paper manuscript.

The research leading to these results has received funding from the Research Council of Norway program IKTPLUSS, under the R&D project “Ars Forensica — Computational Forensics for Large-scale Fraud Detection, Crime Investigation & Prevention”, grant agreement 248 094/O70.

References

Årnes, A., Flaglien, A., Sunde, I.M., Dilijonaite, A., Hamm, J., Sandvik, J.-P., Bjelland, P., Franke, K., Axelsson, S., 2017. Digital Forensics. John Wiley & Sons, Ltd.

Brezinski, D., Killalea, T., 2002. RFC 3227: Guidelines for Evidence Collection and Archiving. RFC 3227. URL. https://rfc-editor.org/rfc/rfc3227.txt.

Chow, J., Pfaff, B., Garfinkel, T., Rosenblum, M., 2005. Shredding your garbage: reducing data lifetime through secure deallocation. 14th USENIX Security Symposium 331–346.

Dunkels, A., Grönvall, B., Voigt, T., 2004. Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Proceedings - Conference on Local Computer Networks. LCN, pp. 455–462.

Hegarty, R.C., Lamb, D.J., Attwood, A., Attwood, A., Attwood, A., 2014. Digital evidence challenges in the internet of Things. In: Dowland, P.S., Furnell, S.M., Ghita, B.V. (Eds.), Proceedings of the Tenth International Network Conference. INC 2014, Plymouth, pp. 163–172.

Hossain, M., Karim, Y.K., Hasan, R.H., 2018. FIF-IoT: a forensic investigation framework for IoT using a public digital ledger. In: Proceedings - 2018 IEEE International Congress on Internet of Things, ICIOT 2018 - Part of the 2018 IEEE World Congress on Services, pp. 33–40.

Klaver, C., 2010. Windows Mobile advanced forensics. Digit. Invest. 6 (3–4), 147–167. https://doi.org/10.1016/j.diin.2010.02.001. URL.

Kumar, V., Oikonomou, G., Tryfonas, T., Page, D., Phillips, I., 2014. Digital investigations for IPv6-based wireless sensor networks. Digit. Invest. 11 (Suppl. 2), S66–S75. https://doi.org/10.1016/j.diin.2014.05.005. URL.

Li, Y., He, J., Huang, N., Chang, C., 2016. On the recoverability of data in android YAFFS2. In: Proceedings - 12th International Conference on Computational Intelligence and Security. CIS, pp. 665–668, 2016.

Minnaard, W., 2014. Out of sight, but not out of mind: traces of nearby devices' wireless transmissions in volatile memory. In: Proceedings of the Digital Forensic Research Conference, DFRWS 2014 EU 11, pp. S104–S111. https://doi.org/10.1016/j.diin.2014.03.013.

Montasari, R., Hill, R., 2019. Next-generation digital forensics: challenges and future paradigms. In: Proceedings of 12th International Conference on Global Security, Safety and Sustainability, ICGS3, 2019.

Pagani, F., Fedorov, O., Balzarotti, D., apr 2019. Introducing the temporal dimension to memory forensics. ACM Transactions on Privacy and Security 22 (2), 1–21. URL. https://dl.acm.org/doi/10.1145/3310355.

Quick, D., Alzaabi, M., 2011. Forensic analysis of the Android file system Yaffs2. In: Proceedings of the 9th Australian Digital Forensics Conference (December), pp. 100–109.

Sandvik, J.-P., Årnes, A., mar 2018. The reliability of clocks as digital evidence under low voltage conditions. Digit. Invest. 24, S10–S17. https://linkinghub.elsevier.com/retrieve/pii/S1742287618300355.

Sandvik, J.-P., Franke, K., Abie, H., Årnes, A., 2021a. Coffee forensics — reconstructing data in IoT devices running Contiki OS. Forensic Sci. Int.: Digit. Invest. 37.

Sandvik, J.-P., Franke, K., Årnes, A., 2021b. Towards a generic approach of quantifying evidence volatility in resource constrained devices. In: Montasari, R., Jahankhani, H., Hill, R., Parkinson, S. (Eds.), Digital Forensic Investigation of Internet of Things (IoT) Devices, Advanced Sciences and Technologies for Security Applications. Advanced Sciences and Technologies for Security Applications. Springer International Publishing, Cham, pp. 21–45. https://doi.org/10.1007/978-3-030-60425-7_2. Ch. 2, link.springer.com/10.1007/978-3-030-60425-7_2.

Sutherland, I., Evans, J., Tryfonas, T., Blyth, A., apr 2008. Acquiring volatile operating system data tools and techniques. ACM SIGOPS - Oper. Syst. Rev. 42 (3), 65–73. URL. https://dl.acm.org/doi/10.1145/1368506.1368516.

Zimmermann, C., Spreitzenbarth, M., Schmitt, S., Freiling, F.C., 2012. Forensic analysis of YAFFS2. In: SICHERHEIT 2012 – Sicherheit, Schutz und Zuverlässigkeit, pp. 59–69.