Towards a generic approach of quantifying evidence volatility in resource constrained devices

Jens-Petter Sandvik[1,2], Katrin Franke[1], and André Årnes[1,3]

[1]Norwegian University of Science and Technology (NTNU), Norway
[2]National Criminal Investigation Service (KRIPOS), Norway
[3]Telenor Group, Norway

July 20, 2020

Abstract

Forensic investigations of the Internet of Things (IoT) is often assumed to be a combination of existing cloud, network, and device forensics. Resource constraints in many of the peripheral things, however, are affecting the volatility of the potential forensic evidence, and evidence dynamics. This represents a major challenge for forensic investigations. In this chapter, we study the dynamics of volatile and non-volatile memory in IoT devices, with the *Contiki* operating system as an example. We present a way forward to quantifying volatility during the evidence identification phase of a forensic investigation. Volatility is expressed as the expected time before potential evidence disappears. This chapter aims to raise awareness and give a deeper understanding of the impact of IoT resource constraints on volatility and the dynamics of forensic evidence. We exemplify in which way volatility can be quantified for a popular operating system and provide a path forward to generalize this approach. The quantification of the volatility of potential evidence helps investigators to prioritize acquisition and examination tasks to maximize the likelihood of collecting relevant evidence from resource-constrained devices. Our work contributes to establishing a scientific base for evidence volatility and evidence dynamics in IoT devices. It strengthens methods for on-scene triage, event reconstruction, and for assessing the reliability of evidence findings.

1 Introduction

As the Internet of Things (IoT) gains traction, the number of criminal cases involving IoT systems is increasing. The increase in IoT ubiquity in all aspects of daily life will extend both the dependence on these systems and increase the number of devices used for crimes. As more IoT systems will sense their environments, they will also act as new sources of evidence for activities in their environment. From these IoT systems, data and information are in a fast flux, and a crime investigator has to prioritize his or her efforts to collect the relevant data as evidence as long as it exists for the criminal case. A formal approach to volatility quantification requires a well-defined terminology of evidence dynamics and volatility. In this chapter, we are defining key concepts and motivating the formal approach, which will be detailed in the remainder of the chapter.

Challenges introduced by IoT systems for digital forensics are abundant [1]. A subset of IoT forensic challenges that affect volatility are summarized as follows: *(i)* The ubiquity of their presence, *(ii)* the resource-constraints, *(iii)* the lack of interfaces for forensic data collection, and *(iv)* the data process flow. The data process flow makes data generated by a device hard to locate and to collect, and it can change the data during its lifetime in the system. The set of data that is collected from the system and is used in the investigation is regarded as *evidence*. The changes to data that will be used as evidence are part of the *evidence dynamics*. Evidence dynamics is a term used for all changes a piece of evidence experiences from the creation of the data to the case has been presented in court [2].

Volatility is a term that describes the time interval before evidence disappears, and the term will be defined in this chapter. The disappearance of evidence is a change that happens to it, and it can thus be seen as a subset of evidence dynamics.

It is not only the IoT devices that are resource-constrained but also forensic investigations are limited by resource constraints. This is a double burden. The resources that are available for an investigation

are finite, and this includes both manpower, time, and equipment. An investigator with access to several possible sources of evidence needs to prioritize between these to optimize the probability of finding the most valuable evidence. This prioritization task is often referred to as *triage*. Roussev et al. defined triage as "[...] a partial forensic examination conducted under (significant) time and resource constraints" [3].

During triage, the investigator needs reliable and objective sources of information to prioritize the data and evidence collection. Given the available resources for the forensic investigation as well as the evidence dynamics, this prioritization is done to maximize the probability of finding relevant data and evidence. Objective and reliable sources of information about the IoT system can help reduce human errors due to cognitive biases a human investigator is susceptible to.

To overcome the misconception that no evidence can be found in resource-constrained peripheral devices, we are aiming to provide an objective measurement to determine the time window where relevant data is most likely to exist, despite its evidence dynamics. This will increase the confidence of the investigator that evidence exists and where it can be found. The knowledge about the likelihood of some evidence is still present in the system after a given time can be of help when prioritizing between collecting data from two different devices that both might contain evidence, but where one has much lower volatility than the other.[1]

This chapter focuses on the volatility of the evidence, and how we can measure it, which is the objective measure as motivated above. Our contributions are:

- A model of the volatility, to better understand the influencing elements of volatility, with the operating system (OS) Contiki as an example.
- The use of statistical tools to measure the volatility, which borrows from the field of dependability and reliability analysis.

The model of the volatility is a construct to split the analysis of the volatility into smaller, well-defined elements that individually contribute to the volatility for the whole IoT system. The statistical method is a quantification of the contributions of both the individual and the combined elements in the volatility model of the IoT system.

Section 2 describes related work in volatility and how this term has been used. Section 3 introduces the concepts of data volatility and information volatility, together with a model for both data volatility and information volatility. Section 4 introduces the use of statistical methods for measuring the volatility. Section 5 uses the Contiki OS as an example of how the model can be used. Section 6 summarizes and concludes this chapter together with discussions on further work.

2 Related work

The research in volatility has been focused on the acquisition process, and how to collect evidence in a forensically sound matter, such that the collection process does not change or otherwise overwrite relevant evidence, maintaining evidence integrity. In the case of such changes happening, the acquisition should minimize the number of changes and document what has changed as a part of the chain of custody. This leads to the concept of *order of volatility* (OOV).

The IETF Request For Comments (RFC) 3227, "Guidelines for Evidence Collection and Archiving", is a best practice guide for collecting and preserving evidence from computer systems [4]. In this guide, the *order of volatility* is listed as an important aspect of evidence collection, as the evidence should preferably be collected from the most volatile evidence, and proceed with the less volatile evidence. The order of volatility thus forms the order for prioritizing evidence collection. Examples of evidence in this RFC, ordered in decreasing volatility order, are registers and cache; routing table, ARP cache, process table, kernel statistics and memory; temporary file systems; disk; remote logging and monitoring; physical configuration and topology; and archival media.

The order of volatility is an assessment tool, and during an investigation, the investigator can decide to rather decrease the risk of overwriting less volatile data on the cost of not collecting more volatile data. As an example, the investigator might want to turn an alarm central off, so that the non-volatile memory won't fill up with warnings, overwriting relevant data from the investigated incident [5].

Ruan and Carthy discuss the order of volatility for cloud providers, and they defined the order of volatility to be, in decreasing order: Service layer artifacts, abstraction layer artifacts, and physical layer

[1] This assessment can go both ways. Either prioritize the high volatility device to collect data before it disappears, or prioritize the low volatility one because there is only time to collect data from one of the devices.

artifacts [6]. This is a more generic model than the one defined in RFC 3227 but covers a variety of system architectures.

Dykstra and Sherman researched available tools and the trust challenges that arise with cloud computing and the forensic collection of data from Infrastructure-as-a-Service solutions [7]. The authors described a model with layers of trust in the system where evidence collected from higher layer abstractions such as applications running in a virtual machine need to trust more of the system than, e.g., packet capture at the hardware level. This idea behind the layers in the system as *layers of trust* is similar to the discussion in this paper on the *storage stack layers*, but in this chapter, we focus on the layers from a volatility perspective.

The trust issue and the "changeability" of data have also been the focus for Casey, where he discusses the need for the forensic examiner to detect, quantify, and to compensate for unforeseen changes to the system caused by errors or loss [2].

Some authors have described the challenges with volatile data in systems. Zulkipli et al. point out that the volatility complexity is higher in IoT systems than in other systems, and they see a need for new techniques for filtering and collecting data in IoT environments [8].

Montasari and Hill also discuss the challenges with volatile data in IoT systems, especially with short-lived data in resource-constrained devices together with cloud aggregation and processing of data in the system [9]. The resource-constriction means less memory and, thereby, more volatile data. The cloud aggregation would lead to challenges in the chain of custody, as it will be harder to track the pathway of the data in the system and describe the changes that have happened to it.

In a paper by Sandvik and Årnes, the volatility of the registers keeping the current clock state under low power conditions was discussed. Testing showed that for some devices, the registers kept the state up to 10 seconds while the processor was connected to a lower voltage than the processor could operate normally under, and it did not have enough power to run the operating system [10]. The results showed the evidence dynamics, as the low power affected registers holding the clock value, which made the clock of the operating system to show the wrong time when power was restored.

3 Volatility in IoT devices

Any stored data will disappear after an amount of time, whether it is stored in electrically powered circuits or engraved on stone tablets. How fast the data disappears is obviously different for these two technologies, and this can be denoted as the volatility of the data. An intuitive attribute of the volatility is that the faster information disappears, the higher the volatility of the data, but to show how volatility should be defined, we need to go into the details of both what we mean with "disappear" and what we mean with "data". Another intuitive attribute is that the volatility is in some way quantifiable and that we, in general, don't know exactly when the data will disappear, so there are probabilities involved.

From the field of information theory, there is a distinction between the terms *information* and *data*. The information source transmits messages that contain some information, and these messages are encoded in data [11, 12]. This terminology is adopted here, and in our case, we can view the information transmitted as the messages about the events that affect the system, and the data is stored in bits in various physical locations of the IoT system.

With this backdrop, *data volatility* is introduced here as the disappearance of data in the system, and the *information volatility* is introduced as the disappearance of the information about an event, or set of events in the system. Figure 1 shows the difference between these two terms. Data volatility only concerns the specific data and copies of that data found in the system. One example of this might be a file that is stored in a device and then copied to other devices in the system automatically. How fast the data content of this file disappears is data volatility. As the information is stored in the system as data, information volatility is dependent on the data volatility and can be viewed as a superset of data volatility. Information volatility takes into account all data that can be used for reconstructing an event in the system. Even though some data that are stored will disappear, there might still be enough information in the system to reconstruct an event. If data is disappearing, there is at one point in time not enough information to reconstruct an event, given the defined certainty, and this is the point where the information about an event has disappeared.

3.1 Data volatility

Data volatility is usually what most people refer to when mentioning volatility. The *order of volatility*, which is commonly used as a reference, is a description of the ordering of data lifetime between the

Figure 1: The difference between data volatility and information volatility. Green signifies existing data, while red signifies inaccessible data. While data volatility focuses on the existence of a specific piece of data, here labeled "Data 1", information volatility focus on all data that can be used for reconstructing an event.

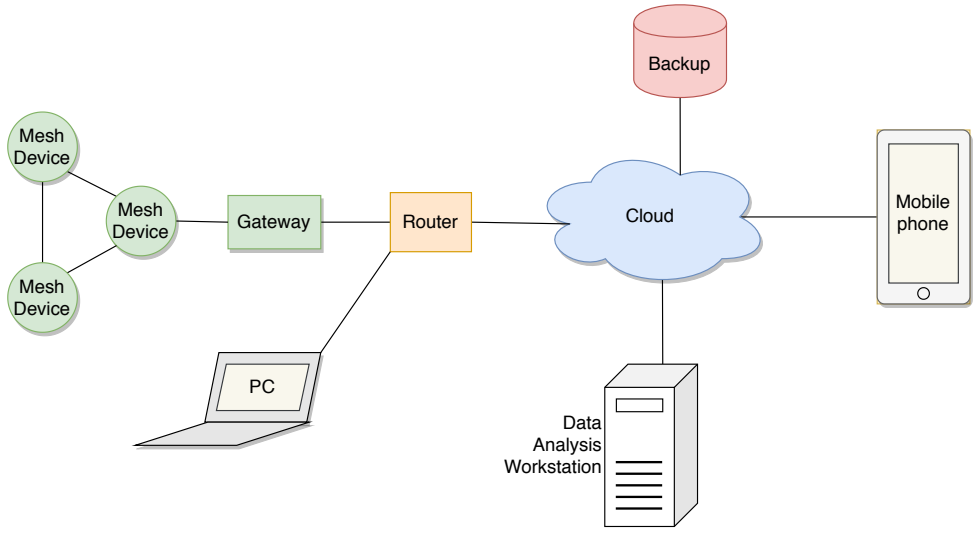

Figure 2: A generic IoT system with data stored in several locations. Each of the elements in the figure have their own internal system for processing and storing the data in question.

various storage types and locations, while the term *volatile memory* is a term used for a type of memory where data disappears as the electric power is removed.

IoT systems can, in many cases, be considered distributed systems, where information about events is stored in many locations. This should also be considered when assessing the volatility of the data. Figure 2 shows a generic IoT system, where data can be cloned into several subsystems and several locations in each subsystem. It is important to establish the data locations that are considered in the volatility model, whether we consider data as it is stored in one location or all copies of the same data as it is stored in the system.

The simplest model is where data is contained in one storage location. This can be a timestamp in the metadata of a file or the contents of a file. The time it takes before the particular data is unavailable for the investigator can be considered the volatility. For this, we need to consider the time it takes before a file, or the data is deleted, and the time it takes before the areas in memory (either volatile or non-volatile) containing the data are overwritten. An example of this might be that deletion and erasure of a file. The information about the file and its content is encoded in data found in both RAM and Flash storage, and while the data disappears from the process memory and file system abstraction layer, the contents are still found in a page in the flash memory.

A more complex model is where copies of the data are found in several locations in the system. A system in this regard can be a single device consisting of CPU, RAM, and flash memory, or it can be a whole IoT system consisting of several IoT devices, servers, routers, cloud storage, and/or computers. Copies of the data can be found in several places in a system, and it is intuitive to think that the storage location with the lowest volatility is the one contributing the most to the overall volatility. If the contents of a file have been overwritten, there is still a probability for the original pages of the file to be located in the flash memory, as the wear leveling algorithm will write to other pages when the file is modified. Even if the pages are erased with a TRIM command, a command for wiping non-allocated blocks in a flash memory device, the data might still be in a page in a bad block, and therefore not erased. The file can also have been copied to other devices or a cloud service.

If we define the data volatility as the time of disappearance of all copies of the data, the probability for finding at least one of the copies of said data is dominated by the storage location with the lowest volatility. The "disappearance" of data is not a sharp boundary between the existence and non-existence of the data. On the one hand, we can think of disappearance as the point where the data does not exist anymore, or is lost to the mythical place together with single socks from the washing machine. The data is erased, and there are no theoretical methods of reconstructing the data from the storage medium. On the other hand, we can view the disappearance of the data as the point where the data becomes inaccessible for the investigator.

The *inaccessibility* of the data requires another set to describe the volatility, namely the methods and tools available for the investigator for acquisition and examination of the data. There are many ways data can be collected from a system, from documenting status indicators on a user interface to

Table 1: The elements of the volatility model.

Model element	Type	Description
Storage abstraction layer, L	Physical	Physical, or close to physical storage layers.
	Logical	Data structures and access methods for the logical storage layer structure in the system.
	Application	The layer that processes the data that are encoded from the events. The application activity functions, A, works on this layer.
Events, E	External	External events affecting the system
	Internal	Internal events in the system, such as delayed response to external events or timed events.
Application activity functions, A	Applications	The applications that handles the data creation, modification and deletion, together with their rates and probability distribution.
Storage management functions, M	FTL	Software and firmware mapping the application storage to physical storage.
	VMM	
Memory device reliability, D	RAM	All devices that contain memory in the system. Each type of memory has its own reliability function and dependencies for failure-free service.
	SSDs	
	Tape drives	
	HDDs	
Environment, S	Configuration	The current configuration of the system.
	Physical environment	The physical environment affecting the system.
	Operational environment	The external operational environment, usage patterns, attack intensity, etc.

desoldering and reading the flash chip or using JTAG to dump the memory. Each of these methods has access to a subset of the data that exists in the system. If the investigator does not have the tools for performing a JTAG acquisition, the data is inaccessible for the investigation. The definitions of *order of volatility* as described in Section 2, often mentions CPU registers and CPU cache as the most volatile storage locations, but there are no practical ways of accessing these for an investigator, as any use of the processor would overwrite the registers and many of the cache lines.

For an investigator, data disappearance means that data becomes inaccessible for the investigation, and we define the term *data disappearance* as data that becomes inaccessible for a given acquisition method or technique. The data will, therefore, have different volatility depending on where the data is collected from in the system, which translates to the acquisition method.

3.2 A model for data volatility

There are many processes and variables in a system that affect volatility. To split the challenge into more manageable parts, we introduce a model to ease the analysis of data volatility.

The generalized data volatility model, $\mathcal{V}_\mathcal{D}$, is introduced here as a 6-tuple given by:

$$\mathcal{V}_\mathcal{D} = (L, E, A, M, D, S) \tag{1}$$

Where L is the various storage system abstraction layers. E is the set of events that has happened in the system, both internally triggered events and external events. A is the functions of the applications producing, modifying and deleting data, M is the functions of the storage management software and firmware mapping the application data to the physical storage devices, and D is the set of individual memory devices in the system with their physical reliability functions. S is the environment of the data storage devices in the system, including the IoT system with its hardware, software, configuration, the physical environment, and the operational environment. Table 1 shows this model with a short description of the elements.

In short, L is the structure of the data pathway in the system; E is the set of events; A, M, D are functions that operate on the data; and S is the environment in the background of the system. The relationship between these elements of the volatility model is shown in Figure 3 and is described in more

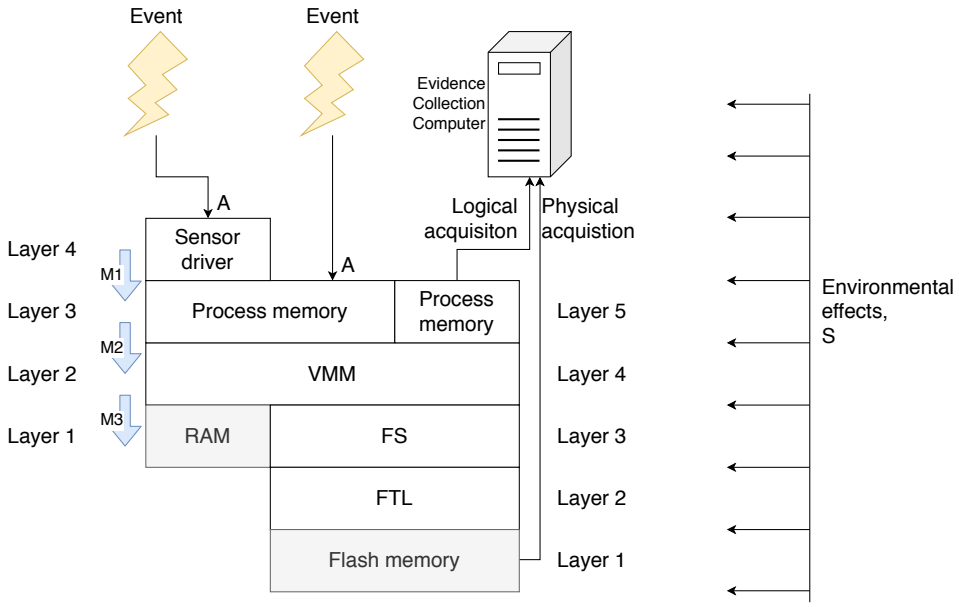

Figure 3: The relation between the elements of the volatility model. The stack is different depending on both the application layer and the physical storage. Note that there is another stack for the data as it is forensically collected, and this varies with collection method.

detail in the following subsections. Each of the elements the storage stack consists of contributes to the volatility of the data. The challenge is to find the amount and type of contributions from each element, so the total volatility of the data can be calculated or approximated if we choose to disregard elements with an insignificant contribution.

3.2.1 Storage abstraction layers

The storage abstraction layers, L, is the set of storage abstraction layers for the devices in the system. This is in the model for two reasons: It makes it possible to analyze the data loss functions from each layer, it is also used for describing which layer the acquisition method uses together with the volatility.

The OSI model of networking protocols describes the abstraction layers for how data can be transmitted over networks, where each layer has a defined role in the addressing and handling of the data packets. This model is not followed to the letter in contemporary network protocols, but it is still a good tool for analyzing network protocols and for learning the abstraction layers in any network protocol stack. A similar structure for storage abstraction layers can be defined.

The storage abstraction model defined in this work specifies how each level in the storage hierarchy handles the storage of information from the application storage layer through the file system abstractions and to the storage in the physical medium. The stack will be different for different applications and physical storage locations, as an application may operate on different levels of the storage stack.

The two layers that are always present are the application layer at the top of the stack and the physical layer at the bottom. The *application layer* is the data that is an interpretation and encoding as a response to events. The encoded data is then stored in a data structure, together with other information, the operating system will keep the process data in other structures, and the data will be stored in the *physical layer*, first in physical RAM, before it can be stored in physical flash storage.

These layers define where data can be forensically collected. Physical acquisition is a term used for forensically collecting data from the physical, or a layer close to the physical layer. Sometimes to forensically collect data from a hardware-near layer, such as the flash translation layer (FTL), is called a pseudo-physical acquisition [13]. The forensic collection of data from any of the other storage layers is often referred to as a logical acquisition, regardless of the exact storage layer that is affected.

Various memory technologies have different names and number of abstraction layers, but at least three layers are consistent among the technologies: The physical layer, the addressable-to-physical translation layer, and the application layer. Table 2 shows examples of a DRAM and a Flash memory layer structure. In RAM, the application stores data in the process memory, and the buffers holding the data can be copied otherwise managed during its lifetime. Beneath the in-process memory handling, the Virtual Memory

Table 2: Example storage abstraction layers and functions for DRAM and Flash memory.

Layer	**DRAM**	**Flash**
Application	Sensor reading	Sensor reading
Translation layers	In-process memory management	File system driver
A2P translation	Virtual Memory Manager	Flash Translation layer
Physical	SDRAM cell	Floating gate transistor

Manager (VMM) handles memory pages and can move the memory pages in and out of a swap file, or other memory modules in case of a NUMA architecture.[2] For the Flash memory layer, the data is first in RAM, and as the file is written to disk, the file system driver will decide where the file is to be written in the address space, and the flash translation layer will move the data from the linear address space to store it in the physical flash pages and keep an index of the corresponding logical address.

It is important to note that the number of layers is dependent on several factors: How many processes or functions are handling these data and the number of abstraction layers within one architecture. The data producer stack can also be different from the collection stack for the same physical data. This can, e.g., be when an IoT device does write directly to flash, but the interface for collection is to connect a computer to the device and logically acquire the data via en Media Transfer Protocol (MTP), which adds a layer of abstraction that the writing did not go through.

3.2.2 Events

The events, E, is the set of events affecting the system, often initiated by some external interface to the system, as an IoT system is typically an open system. Events originating in the system can be events triggered by a timer or triggered by a state change of the system. The events are often what an investigation tries to reconstruct from the stored data. External events are events that are triggered from outside the system, while internal events are events originating from the system. The events get encoded into data in the storage abstraction stack and end up in one or more physical storage locations.

3.2.3 Application activity function

The application activity function, A, is the set of functions that processes the information and translate events into the storage system. It handles creation, modification, and deletion of data from the top layer of the storage abstraction layer stack: $A : E \rightarrow L_{\text{top}}$. One event can trigger several of these functions. An example of an application activity function is a program running on an IoT device, analyzing sensor inputs, recording sudden changes, and deletes the data after one week.

As the application layer is the data encoding of the external inputs generated by an event, this layer will always be present, and the application activity function will always map events into the application layer. An example if this is log rotation, where the oldest log file is deleted while a new one is created, and the other log files are renamed. Another example is the reading sensor inputs and processing these values. There might also be unexpected events, such as a sudden power failure that affects RAM contents, and non-volatile content that is in the middle of a non-atomic write operation.

3.2.4 Storage management functions

The storage management functions, M, are the functions mapping the data between the intermediate layers in the storage abstraction stack: $M : L_x \rightarrow L_y$. One example of this is the flash translation layer, which reorganizes the logical storage address to the flash memory pages. These functions can copy data between locations in a lower layer transparent for the layer above. From the application's view, there is only one occurrence of the data, but it may exist at several physical locations. The deletion of data from the application will also make the data inaccessible for the application layer, but the data might still exist in the physical medium.

The pathway for the data between the application layer and the physical layer can be different depending on the application, the data, and the physical layer, so the storage management functions will not necessarily be the same for all data in a system.

[2] Non-Uniform Memory Access.

3.2.5 Memory device reliability

The memory devices failure probability, M, is the failure function of the physical storage locations of the data. Hardware failures of a memory device will render a part or all of the data inaccessible and thus impact volatility. We can define at least two different aspects of this. The first is the reliability under normal operation; the other is the reliability after an event in the system affecting the memory device has happened. An example of this is the clock registers when losing power. Sandvik and Årnes reported a retention time for the value in the clock register of up to 10 seconds after an abrupt power loss [10].

Typically, each device will have a failure probability distribution or reliability function. This operates on the lowest level of the storage abstraction layer stack and gives the volatility for the stored data in the absence of other events affecting the data.

3.2.6 Environment

The environment of the system, S is the parts of the system environment that can affect the volatility of the data apart from the direct events. The physical environment can impact the lifetime of the components; the radio environment can affect communications; the digital operating environment shows the attack base rate, or how hostile the digital environment is. The environment can change from one state to another or show cyclical changes over time.

3.3 Information volatility

From an investigator's point of view, the data loss in a system is not the most critical part of the investigation in itself, but rather that the amount of information available should be enough to reconstruct events with a given confidence. While some data might disappear, there might be other data that can be used for reconstructing the same events. Information about an event is often encoded by several application functions, spread over many pieces of data, and stored in several locations, as shown in Figure 1.

Information volatility is the probability for enough data to be present to reconstruct events after a period. As data is needed for decoding the information, the data volatility, as discussed, is an important part of, and can be considered a subset of information volatility.

A model for information volatility can be defined as:

$$\mathcal{V}_{\mathcal{I}} = (\mathcal{V}_{\mathcal{D}}, T, C) \tag{2}$$

Where $\mathcal{V}_{\mathcal{D}}$ is the set of data that the event is decoded into, T is the threshold for the certainty or confidence, that is needed to reconstruct the event, and C is the decoding function that interprets the data into information about the event.

3.4 Forensic resources

The volatility model, as described, focuses on the technical part of data and information volatility. For this to be relevant for an investigator, we also have to comment on the socio-technical perspective that is the human and organizational aspects of the investigation. The resource-constraints to the investigation itself is a burden that adds to the resource-constraints of the devices in an IoT system.

To get access to the data or information stored, the investigator is dependent on resources. This can be both personnel, time, acquisition tools, storage space, or knowledge. The available resources affect both the amount and the quality of the collected and examined data. The equipment and knowledge decide the type of acquisition that can be performed, which again decides which memory devices that can be acquired and the layer of the storage abstraction stack that the data can be acquired from. As the volatility of data has to be assessed based on the storage layer from which it is collected, the available resources do affect the perceived accessibility and, thus, the perceived volatility of the evidence. The resources are, however, not a part of the model, but is considered a part of the limitations to the investigation.

4 A statistical approach to data volatility

The term *volatility* has so far been used to describe the time interval before some data becomes inaccessible to the investigator. To quantify the volatility so it can be used for predicting the probability of finding

relevant evidence after a time, a statistical model is needed. We will borrow some methods from reliability and dependability analysis.

We identify that the reliability of a system, or the probability that the system fails within a given period, is similar to the concept of volatility, where the volatility can be viewed as the probability for the evidence to become inaccessible within a given period. The volatility is thus the *reliability of data*, where the reliability analysis' concept of non-repairable failure is the volatility's data inaccessibility. As discussed in Section 3, the volatility is dependent on the particular component in the storage stack in which the evidence collection takes place. This means that the probability distribution can vary, depending on the acquisition method, and a volatility function is valid for a particular element in the storage stack.

The reliability function, $R(t)$, is a function that describes the probability that the system has not failed at time t. For a steady state system the reliability function is given by: $R(t) = P(T_F > t)$, where T_F is the time to failure. Instead of *time to failure*, we can use the term *time to inaccessibility* for data and define a volatility function as the probability of the data being inaccessible at time t:

$$V(t) = P(T_I > t) \tag{3}$$

where T_I is the time to inaccessibility. The *mean time to inaccessibility* (MTTI) can then be expressed similar to the *mean time to failure* (MTTF) as:

$$MTTI = \int_0^\infty V(t)\, dt \tag{4}$$

The actual distribution of the volatility function is not generally known, as many variables affect the exact distribution, such as the application's memory and file system operation distribution, encryption, the allocation strategy of the file system, artifacts of the physical storage medium, reboots, or system failures. The model in Section 3 does, however, give us some idea about the contributions to the volatility from the various parts of the system. When the data is deleted from the application is dependent on the application function, and the physical layer is dependent on the reliability of the memory chip. The intermediate management layers can also hide or copy data, as we saw in the model description.

To model the volatility, we have to find the corresponding probability distribution function (PDF). Several distributions are used in reliability analysis, and these are candidates for volatility analysis. The distribution can be estimated by empirical testing of the system and matching the PDF, and analytically by assessing all contributing factors to the volatility. Two commonly used PDFs are the Exponential distribution and the Weibull distribution, the former is popular because of its simplicity, the latter because of its flexibility. The distributions and are described in more detail below, together with the motivation of using them.

4.1 Exponential distribution

The exponential probability distribution is a simple probability function that is popular because of its simplicity but is not always a good approximation of the reliability function [14]. The exponential distribution works well for independent events, has a constant intensity, and a memoryless property, which can be true for external events in the volatility model described in Section 3.2.2. The probability distribution function has one parameter, λ, which is the rate of the data deletion. The distribution is given by:

$$f(t, \lambda) = \lambda e^{-\lambda t} \tag{5}$$

The cumulative distribution function is given by:

$$F(t) = 1 - e^{\lambda t} \tag{6}$$

Figure 4 show examples of the PDF and the CDF for the exponential distribution for various values of λ. A λ of 0.1 means a rate of 0.1 events per unit of time, and 0.01 means 0.01 events per unit of time. This can be, e.g., an internal event like a garbage collection routine, happening every 100 seconds on average, which gives a λ of 0.01 events/s. The mean of this distribution is λ^{-1} and the reliability function for this distribution is given by $R(t) = e^{-\lambda t}$. This can be a good description for the events encoded by the application activity function, as external events might be modeled as happening at a constant intensity, triggering modifications and erasure of data in the file system.

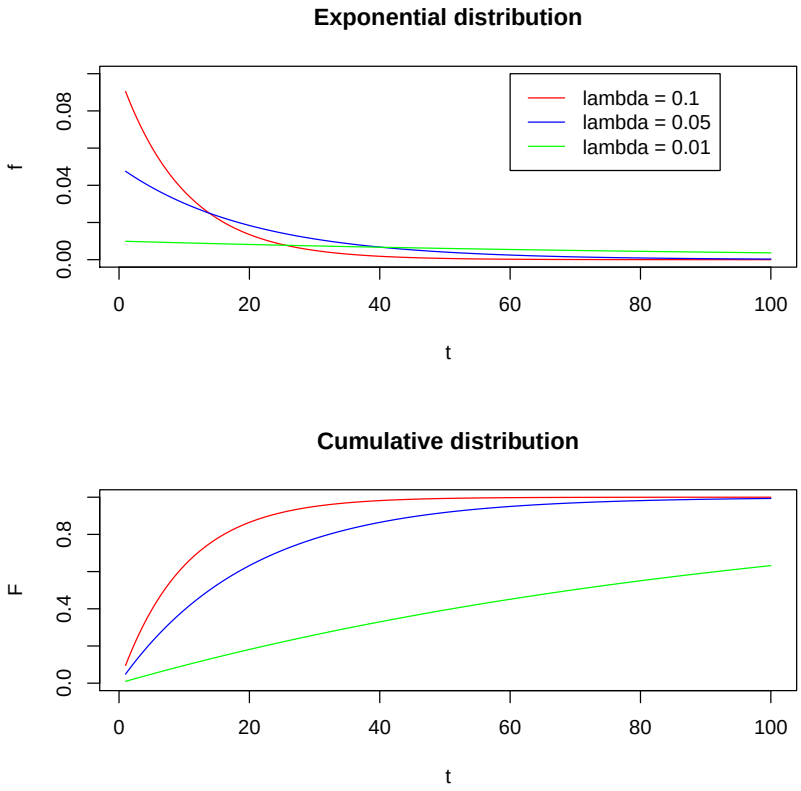

Figure 4: Examples of the exponential probability and cumulative distribution functions.

4.2 Weibull distribution

The Weibull distribution is another common distribution used for modeling reliability [15]. It has two parameters that can adjust the shape of the distribution, α and γ, which are called the scale parameter and the shape parameter, respectively. The flexibility of the function lets it approximate several other distributions. Depending on the shape parameter, it can model falling ($\gamma < 1$), steady ($\gamma = 1$), or rising ($\gamma > 1$) failure rates.

This distribution is often used to model physical components' failure rate, as the failure from wear of the components is not constant over time, but changes with the age of the component. Flash memory in SSD storage devices is an example of a failure distribution closely resembling a Weibull distribution [16]. This can, therefore, fit the physical reliability function in the volatility model, as described in Section 3.2.2.

The probability distribution can be parameterized in several ways, two of them are given below:

$$f(t, \gamma, \alpha) = \frac{\gamma}{t}\left(\frac{t}{\alpha}\right)^{\gamma} e^{-\left(\frac{t}{\alpha}\right)^{\gamma}} \tag{7}$$

$$f(t, k, \lambda) = k\lambda(\lambda t)^{k-1} e^{-(\lambda t)^{k}} \tag{8}$$

When $\gamma = 1$, this distribution is identical to the exponential distribution, with $\lambda = \alpha^{-1}$.

The cumulative distribution function is given by:

$$F(t, \gamma, \alpha) = 1 - e^{\left(\frac{t}{\alpha}\right)^{\gamma}} \tag{9}$$

Figure 5 shows examples of the Weibull distribution and the cumulative distribution function for a few values of γ and α. For $\gamma = 1$, the plot is equal to an exponential distribution. As the shape and scale parameters can't easily be decided analytically, the parameters often are estimated by empirical observations and the probability distribution fitted to the data. See also Section 4.4. The mean of the distribution is given by $\alpha\Gamma\left(1 + \gamma^{-1}\right)$, where $\Gamma(x)$ is the Gamma function, which for natural numbers is $\Gamma(N) = (N-1)!$, and a slightly more complex definition for non-natural numbers: $\Gamma(z) = \int_0^{\infty} x^{z-1} e^{-x}\, dx$, where $z \in \mathbb{C}$ and $\Re(z) > 0$. The reliability function for this distribution is given by $R(t) = e^{-\left(\frac{t}{\alpha}\right)^{\gamma}}$.

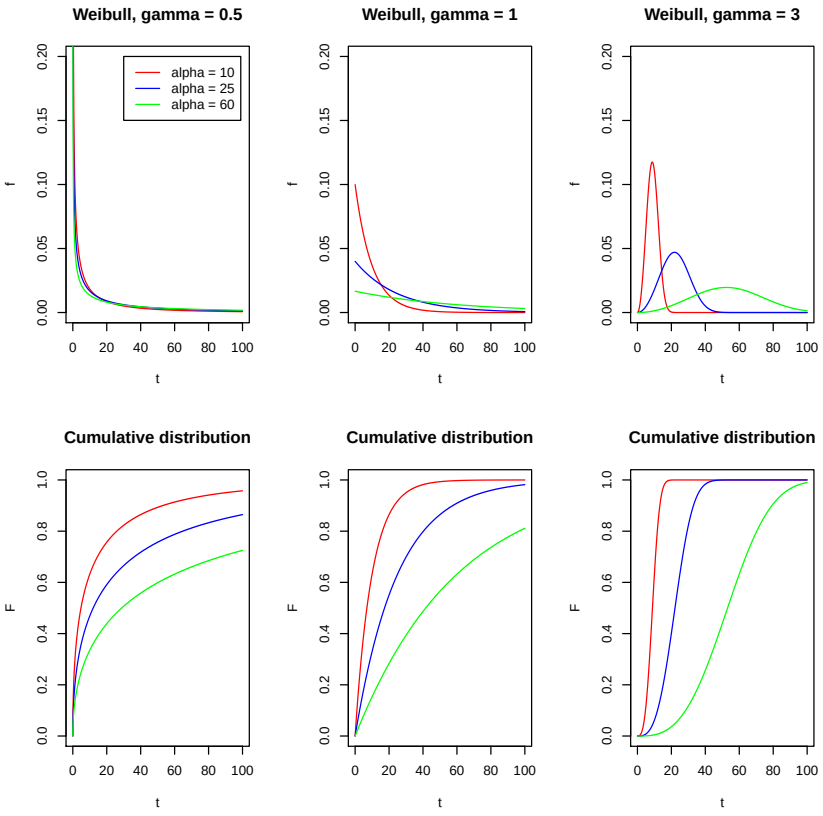

Figure 5: Examples of the Weibull distribution and cumulative distribution functions for some values of γ and α.

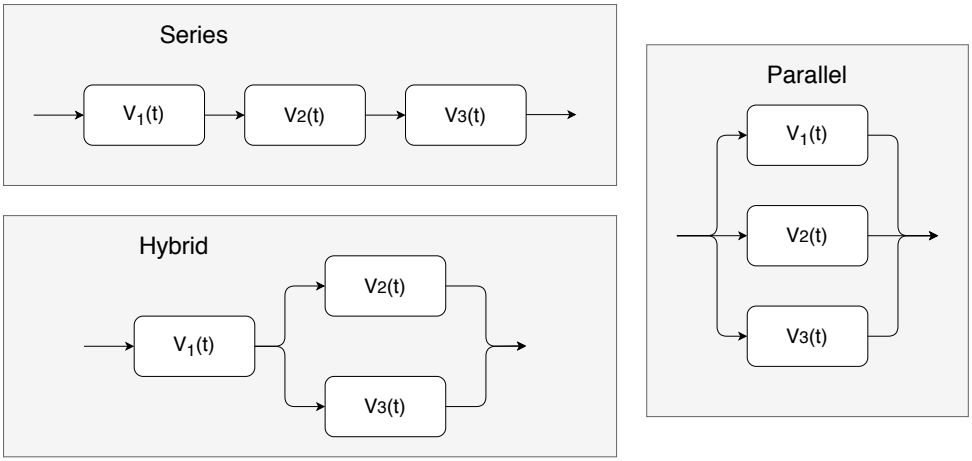

Figure 6: Components in series, in parallel, and hybrid connection.

4.3 Series and parallel systems

The storage system can be seen as a system consisting of elements connected both in series and parallel. This is similar to the block model used for calculating reliability in a compound system [15]. The system can be viewed as such a block model to ease the analysis. Figure 6 shows such a block model of components attached in series and parallel. The volatility function for these connections is given by:

$$V_{\text{series}}(t) = \prod_i V_i(t) \tag{10}$$

$$V_{\text{parallel}}(t) = 1 - \prod_i (1 - V_i(t)) \tag{11}$$

The storage management functions are, in most cases, connected in series. If data is duplicated, there will be a parallel structure in the data pathway. The system's volatility can thus be calculated by combining equations 10 and 11 following the block structure of the system components.

4.4 Probability distribution fitting

Finding a probability distribution and the associated parameters that best fit the observed data, and measuring the goodness of fit, is known as probability distribution fitting. The fitting process should ensure that the model closely fits the observations and that the model selection can be explained.

Distribution fitting can be seen as two different tasks. The first is to find the optimal parameters for a given distribution that matches the observed data, and the other is determining how good the distribution fits the observed data.

Several software packages can help to fit distributions and select optimal parameters. One such software package is the library *fitdistrplus* for the statistical computation and graphics software *R*.[3] Both R and the fitdistrplus library is free[4] software. Sagemath, Matlab, and Mathematica also have distribution fitting functionality.

To measure the goodness of the fit, the observed data has to be compared to the hypothesized distribution, and a test of this hypothesis, that the hypothesized distribution explains the observed data is therefore needed. There are several tests for this, each with their assumptions about the data being observed. Among the tests often encountered, are χ^2-test for discrete data and Kolmogorov-Smirnov test.

5 Example: Contiki-NG

As an example of the storage stack of a resource-constrained IoT device, we can use the Contiki operating system. Contiki and its successor, Contiki-NG, is an operating system for resource-constrained IoT devices [17]. Contiki is built around an event-driven kernel, and utilize loadable modules and services. The whole operating system is about 100 kB, and need at least 10 kB of RAM to run.[5] According to Eclipse

[3] `https://www.r-project.org`, visited 2020-07-01.
[4] Free as in beer and speech.
[5] `https://github.com/contiki-ng/contiki-ng/wiki`, visited 2020-07-08.

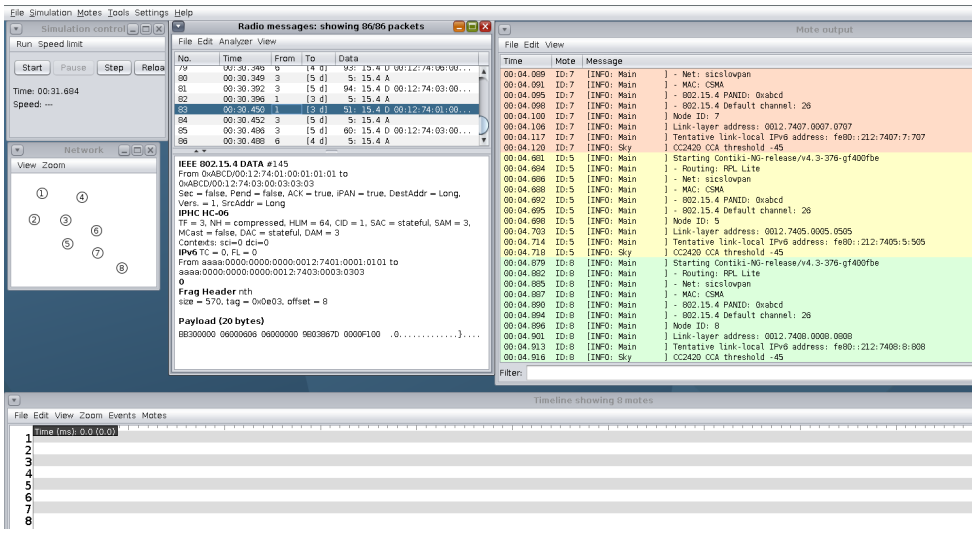

Figure 7: Cooja running a simulation.

Foundation's annual developer survey, about 5 % of the IoT developers were using the Contiki operating system for their projects [18].

For embedded devices, the acquisition of running RAM can be a challenge in investigations, but non-volatile memory is easier to acquire, as the contents still are present after the power has been removed. The non-volatile storage is managed by the Coffee File System, a file system that is both minimalist and designed for flash memory devices. This section, therefore, focuses on the file-system specific part of Contiki. From a volatility perspective, the theory is similar for RAM data, but the specific memory allocation methods need to be taken into account.

Some advantages from a research perspective of using Contiki as a case study is that the Cooja simulator that comes with the OS can simulate various types of networks and configurations and also dynamic environments. It can both emulate specific micro-controllers and run native code on the host architecture. Contiki/Cooja also implements 6LoWPAN and other protocols that are used in IoT systems. Figure 7 shows a screenshot of a running simulation.

The Coffee file system has been designed to run on resource-constrained nodes and to include wear-leveling techniques. Because of the resource constraints, the file system has simplified many operations that we take for granted in a general-purpose file system. It does not contain much metadata, the actual file size is not among the metadata but has to be calculated, and there are no timestamps among the metadata.

Flash memory works differently than old-fashioned hard disks or RAM. Writing to flash memory can only be done by writing a whole page [5]. The page size is specific to a particular chip and is often 512, 1024, or 2048 bytes. A bit can only be set or flipped from '1' to '0', and to flip the bit back to '1', a whole erase block has to be erased. Erase-blocks consist of several pages. This means that modifying data in flash memory involves writing new versions of the data rather than overwriting existing data. A file system operating on a flash memory need to either take this into account or introduce a flash translation layer that mimics an addressable read-write memory area for the operating system, while it hides the data shuffling happening in the background. The coffee file system is designed for operating on a flash device and does not use a flash translation layer.

When a file is allocated, the default is to allocate 11 pages for the file, and when a page in the file is modified, a log file is created, recording a number of changes in the file until there are no more pages left in the log file. The next modification will then trigger a new file with the same name to be created. The default number of updates in the log file is four pages, and the default page size is 0x100 bytes, which equals 256 bytes.

Appending data to the end of an existing file does not create an entry in the log file. As an append operation doesn't modify existing data, the data can be written directly to the already allocated pages. When an append operation has reached the initial file allocation size, a new copy of the file will be made, and twice the number of pages will be allocated.

The erasure of data happens when the write-pointer in the file system reaches the end of the addressable flash memory. This triggers a garbage collection, where all free erase blocks, or sectors as it is named

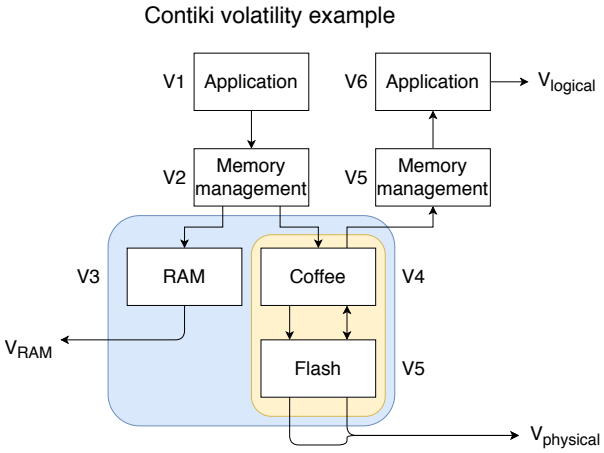

Figure 8: A simplified model of an application running on a Contiki device and 3 different acquisition methods.

in Coffee, are erased and are free to be reused. The file system does not move existing data to free up more pages.

The simplicity of both the file system and Contiki as an operating system removes many of the storage abstraction layers and simplifies the model considerably. From the top, the device has an application running, where it receives inputs, and reacts to this. The application function acts on the events; as an example, it can store a value that is read from a device interface in a file. As the file containing these records is sent to a central server by an application, the log file is deleted by the application. This all happens at the top layer in the stack. Beneath this layer, the data is held in data structures in RAM and temporarily stored in the physical RAM. The file write will open a new stack toward the flash memory and write the files there.

The storage management functions are different between RAM and Flash, and the function will trigger both copies of the data to be spread over the physical Flash memory, and keep track of the unused memory such that new writes can be done at the right location. In the physical flash, the data can be held for a long time, until the flash memory chip breaks, or the flash cell stops working. This is the domain of the memory device's failure function.

Figure 8 demonstrates a simplified block model of an app running on a Contiki device. To calculate the volatility of the system, we can use the equations from Section 4.3 to calculate the volatility for the acquisition methods:

$$V_{\text{physical}}(t) = V_1(t) \times V_2(t) \times V_4(t) \times (1 - (1 - V_{5,1}(t))(1 - V_{5,2}(t))) \tag{12}$$

$$V_{\text{RAM}}(t) = V_1(t) \times V_2(t) \times V_3(t) \tag{13}$$

$$V_{\text{logical}}(t) = V_1(t) \times V_2(t) \times V_4(t) \times V_5(t) \times V_6(t) \tag{14}$$

Each of the individual volatility functions in Figure 8 has to be assessed, to find $V_i(t)$ used in the above calculation. In our example, the data was duplicated when stored in flash memory, therefore the parallel combination in V_5. For the logical acquisition, $V_{\text{logical}}(t)$, the duplicated data in the flash memory is not visible, and not used by the file system when reading, so the whole pathway is in series.

Another storage stack includes the transfer of the file to a central server. Here the storage management functions translate the data to network packets that are sent and stored in RAM of the gateway and other network equipment until it reaches the central server. This is outside the bounds of the Coffee file system, though.

To acquire data, the investigator can perform a logical acquisition, some devices allow the connection through USB, and might use a Media Transfer Protocol for transferring a subset of the files in the operating system. This is shown in Figure 3. The storage stack is different between the application layer that process events and the application layer for the MTP server that transfer the data to the forensic collection computer. If the investigator instead uses a chip-off method, the data will be collected from the physical flash storage layer, at level 1.

6 Summary and conclusions

In this chapter, a model of data and information volatility is introduced. This volatility model is used to analyze the elements in IoT systems that affect the volatility of data and of information in the IoT system. The emphasis in this chapter is on the data volatility. The data volatility model consists of *(i)* an abstraction for a storage layer stack; *(ii)* events and system states affecting data; *(iii)* and functions for how data is transformed between the application and the physical storage. From this data volatility model, all individual components contribute to the overall volatility. Based on the volatility model, we also introduce a quantitative measure for volatility. It allows for a more objective assessment of volatility. The volatility measure is dependent on the forensic method used, as each forensic method collects data from different components in the storage layer stack. We show that the IoT system can have several volatility measures, one for each element in the storage stack used to collect forensic data.

We derive a statistical measurement method from dependability and reliability analysis. This statistical measuring method is used to calculate the volatility contribution in each storage layer component. Quantifying the volatility can thus be established by combining the probability distributions by the data pathway connections in the storage stack. The data pathways can be a mix of series and parallel connections. For the volatility function, two commonly used probability distributions are described, and the *Mean Time To Inaccessibility* is introduced. We have modeled this after the Mean Time To Failure used in reliability analysis.

The model is exemplified using *Contiki-NG*, an open-source, minimalist, and real-time operating system for resource-constrained IoT devices. By using Contiki-NG as an example, we can focus on the core model without the added complexities of more advanced operating systems and storage management functions. In addition, Contiki contains a powerful simulator that we can use for our study and for volatility analysis.

Our study described in this chapter is a step towards establishing a scientific base for measuring the data and information volatility in an IoT system. As IoT systems, in particular, and other computer networks in general, become more and more complex, an objective and reliable assessment of the systems' volatility is becoming more crucial. It works toward a forensically sound acquisition of data with a high evidential value.

With our study, we aim to establish a theoretical foundation toward a scientific base for volatility analysis. Further empirical studies are needed to reveal each element's specific volatility contributions in the volatility model, and to translate this theoretical model into working procedures for forensic practitioners. The similarities and differences between IoT systems and individual devices regarding the components and their volatilities are open for further studies. Information volatility is introduced in this chapter but deserves to be focused in more detail elsewhere.

Our objective is that this research will leverage better tools for the forensic investigator and the forensic community at large, to objectively and reliably be able to quickly prioritize the data collection during the triage process such that the quality of the investigation can be upheld. In the courtroom, enough evidence with high quality is a necessity for a fair trial. By enabling the investigator to collect and analyze more relevant data, the court has a better understanding of the facts to make a just judgment, decreasing the chance of a miscarriage of justice.

Acknowledgment

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 248094/O70.

References

[1] Mauro Conti, Ali Dehghantanha, Katrin Franke, and Steve Watson. Internet of Things Security and Forensics: Challenges and Opportunities. *Future Generation Computer Systems*, 2017.

[2] Eoghan Casey. Error, Uncertainty, and Loss in Digital Evidence. *International Journal of Digital Evidence*, 1(2):45, 2002.

[3] Vassil Roussev, Candice Quates, and Robert Martell. Real-time digital forensics and triage. *Digital Investigation*, 10(2):158–167, 2013.

[4] Dominique Brezinski and Tom Killalea. RFC 3227: Guidelines for Evidence Collection and Archiving. RFC 3227, 2002.

[5] André Årnes, Anders Flaglien, Inger Marie Sunde, Ausra Dilijonaite, Jeff Hamm, Jens-Petter Sandvik, Petter Bjelland, Katrin Franke, and Stefan Axelsson. *Digital Forensics*. John Wiley & Sons, Ltd, 2017.

[6] Keyun Ruan and Joe Carthy. Cloud Forensic Maturity Model. pages 22–41, 2013.

[7] Josiah Dykstra and Alan T. Sherman. Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation*, 9(SUPPL.), 2012.

[8] Nurul Huda Nik Zulkipli, Ahmed Alenezi, Gary B. Wills, N.H.N. Zulkipli, Ahmed Alenezi, and G.B. Wills. IoT Forensic: Bridging the Challenges in Digital Forensic and the Internet of Things. In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security*, number January, pages 315–324, 2017.

[9] Reza Montasari and Richard Hill. Next-Generation Digital Forensics: Challenges and Future Paradigms. *Proceedings of 12th International Conference on Global Security, Safety and Sustainability, ICGS3 2019*, 2019.

[10] Jens-Petter Sandvik and André Årnes. The reliability of clocks as digital evidence under low voltage conditions. *Digital Investigation*, 24:S10–S17, mar 2018.

[11] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, jul 1948.

[12] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2 edition, 2006.

[13] C. Klaver. Windows Mobile advanced forensics. *Digital Investigation*, 6(3-4):147–167, 2010.

[14] Kenneth E. Murphy, Charles M. Carter, and Steven O. Brown. The exponential distribution: The good, the bad and the ugly. A practical guide to its implementation. *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 550–555, 2002.

[15] Roy Billinton and Ronald N. Allan. *Reliability Evaluation of Engineering Systems*. Springer US, Boston, MA, 1992.

[16] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. *Performance Evaluation Review*, 43(1):177–190, 2015.

[17] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - A lightweight and flexible operating system for tiny networked sensors. *Proceedings - Conference on Local Computer Networks, LCN*, pages 455–462, 2004.

[18] Eclipse Foundation. Eclipse IoT Developer Survey 2019. Technical Report April, Eclipse Foundation, 2019.