Digital Investigation 14 (2015) S45–S54

Contents lists available at ScienceDirect

Digital Investigation

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

DFRWS 2015 USA

Wirespeed: Extending the AFF4 forensic container format for scalable acquisition and live analysis

Bradley L. Schatz

Schatz Forensic, Level 10 149 Wickham Tce Brisbane, QLD 4000, Australia

Keywords:
Digital forensics
Evidence containers
Live forensics
Acquisition
Imaging
AFF4

A B S T R A C T

Current approaches to forensic acquisition are failing to scale to large devices and fast storage interfaces. The research described in this paper identifies limitations in current widely deployed forensic image formats which limit both the ability to acquire evidence at maximal rates, and to undertake live analysis in today's environment. Extensions to the AFF4 forensic file format are proposed which address these limitations. The proposals have been implemented and proof of concept demonstrated by demonstrating that non-linear partial images may be taken at rates that exceed current physical acquisition approaches, and by demonstrating linear acquisition at rates significantly exceeding current approaches: in the range of 400 MB/s–500 MB/s (24–30 GB/min).

Introduction

Within the field of digital forensics the volume problem is well recognized: more devices and larger storage compound the amount of data to preserve and analyse per case. In 2015, storage technology exhibits a significant amount of diversity: the capacity of spinning disk hard drives grows at a rate faster than I/O rates; SSD's are capable of I/O rates many times faster than their spinning disk cousins; and storage is increasingly absent a direct attachment, located across comparatively low I/O rate networks, such as cloud based storage.

Forensic acquisition has failed to scale with this growth in both volume and I/O rates. In practice, forensic imaging remains generally reliant on the imaging approach defined over a decade ago – the linear and complete image, integrity protected by a hash, and optionally block compressed (Rosen, 2002). In the field, the dominant methods of imaging generally achieve I/O rates in the realm of 100–150 MB/s, with hardware imaging tool manufacturers beginning to promise speeds of a maximum of 250 MB/s (Tableau, 2014). Few empirical studies exist in the literature in regard to acquisition throughput. Zimmerman (2013) observes far lower rates of in the low 100's of MB/s, and Bertasi and Zago (2013) observes peak rates of 110 MB/s.

Source devices with I/O rates exceeding the commodity SATA hard drive threshold of around 200 MB/s are increasingly common. Formerly it may have only been servers with RAID subsystems which produced this load; today's SSD's, and in-cloud virtual attached storage commonly double this rate. Acquisition of such devices commonly proceeds at rates far lower than the maximum I/O rate of such devices.

The above trends result in bandwidth constrained spinning disks longer acquisition times, and in the bandwidth rich devices, sub optimal acquisition durations. Both of these contribute to latency between the identification of target evidential devices and the gaining of meaningful analytic results, due in part to lack of cohesion between forensic processing steps (Roussev et al., 2013).

Current responses to minimize this latency when facing very large sets of evidence fail to be wholly satisfactory to the practitioner: preserve a limited subset of available evidence and run the risk of not preserving potentially relevant evidence (triage), or cause significant availability impacts on computing resources while complete imaging

E-mail address: bradley@wirespeed.io.

http://dx.doi.org/10.1016/j.diin.2015.05.016

of everything proceeds. Current research proposals addressing this end to end latency issue are scarce. The LOTA (Roussev et al., 2013) proposal goes far in conceptualizing the problem as a real time processing problem, however, due to the requirement for low hundreds of CPU cores per target device being processed, the proposed approach is generally only applicable to in-lab usage, limiting its application to evidence seized (collected) from the field and processed within the laboratory.

The primary hypotheses of this research are that latency between the beginning of acquisition and the obtaining of analysis results can be reduced by reframing acquisition as a (potentially) partial and non-linear activity, and by employing a forensic image format that enables the former while not creating bottlenecks.

We posit that for contexts where seizure is not an option (for example Civil, covert, Incident Response, and at scale of hundreds of machines) this latency can be minimized in two ways. First, by employing a forensic container format that enables acquisition to proceed at maximal I/O rates. Secondly, by re-conceptualizing forensic acquisition as a process which occurs hand in hand with live analysis. Computationally feasible analysis tasks, and live analysis tasks which traditionally fall in the domain of triage activities or live preview, are applied as a priority, contributing to the incremental building of a partial (or eventually full) physical image.

Improvements to the AFF4 forensic container which resolve the identified shortcomings are proposed, and their effectiveness evaluated through a systematic set of experiments focused on high input rate acquisition.

Motivation

This section describes at a high level our acquisition and live analysis system (which we call Wirespeed), the implementation and testing of which has identified the shortcomings in existing forensic containers described later in this paper. We note that the architecture of the Wirespeed system is the subject of a separate publication.

Acquisition as an interactive & batch process

Fundamental to our proposal is the idea that the scope of acquisition should be subject to change and prioritized based on live analysis activities of the examiner, and more generally by categories of forensic data identified for acquisition. Starting with an empty forensic image, the image is successively built from target blocks, first of volume metadata and then filesystem metadata. A non-linear, partial image is foundational to such an approach.

Blocks are then preserved into the image based on two separate activities.

An analyst might conduct a live analysis of the storage device via a virtual disk (i.e. iSCSI) interface to the image – blocks present in the partial image are returned to the analyst's preferred tool directly via the virtual hard disk, and blocks not present in the image are scheduled for reading from the target device, and when read, stored in the image and returned to the analyst's tool via the virtual hard disk. These interactive accesses are prioritized by a scheduler ahead of other accesses.

The analyst might also task the acquisition of various categories of data via a separate GUI controlling the acquisition process: our current implementation allows for file subsets such as event logs, registries & page files, and wider categories such as all allocated data, and unallocated data. This enables the analyst to successively widen the scope of acquisition as their live analysis activities proceed. Further, live analysis might result in the determination that the target device is not relevant, and the acquisition process terminated, the resulting partial image remaining as a basis for reproducing the said analysis.

Maximizing acquisition rate

Maximizing the rate at which the above non-linear sequence of blocks might be assembled into a forensic image is a systems problem involving a range of variables, including:

- The maximal I/O rate of the target device;
- The maximal I/O rate of the target device interface;
- The method of hashing;
- The number of cores and CPU speed;
- The method of compression;
- The path taken in reading disk blocks;
- The available I/O interconnects to the image storage device;
- The maximal write rate of the filesystem implementation
- The maximal write rate of the evidence container device.

Depending on budgetary constraints, it is reasonable to believe that on a small scale the examiner may be able to control the majority of these. For example an analyst employing a removed dead disk acquisition using a forensic media duplicator (“drive cloner”) in the field has control over many of the above but the maximal I/O rates of the target device and its storage interface.

In the field and at a scale of numerous devices, examiners are regularly faced with far less control over these variables: budgetary constraints will limit the number of forensic duplicators able to copy removed dead disks in parallel. Common responses to these pressures are the usage of forensic live-CD's to employ in-situ dead disk techniques, necessitating the usage of the CPU and I/O channels of the target system. For such approaches, combined with old server hardware, acquiring via USB2 (max 40 MB/s) is, while not desirable, still in 2015 a reasonable option.

It is well understood in the literature that for at least spinning disks, the I/O rate of reads from the disk is maximal for reads of long runs of consecutive blocks, with seeks imposing a significant penalty in throughput. Accordingly, we adopt a priority based scheduling approach that prioritises interactive analysis over non-interactive acquisition goals, and scheduling the reading of blocks and caching in RAM to minimize disk seeks in acquiring non-linear block sequences.

What is a disk image?

The preceding sections outlined our approach to minimizing latency in a combined forensic acquisition and live analysis system, and posited that the key enabler of the approach is in addressing limitations in the current conception of the evidence container. This section, and the following review the current state of the art in such.

For the purposes of this paper, an image refers to a physical forensic image, which is, by consensus in the field, generally agreed to refer to a near-complete contiguous bit for bit ordered copy of a storage medium, stored with a linear hash of the same bitstream. An image may be split into blocks and optionally compressed in such a form that it supports efficient random access. We say near-complete as the image may differ from the actual source evidence due, for example, to inaccessible sectors yielding discontinuities containing unknown data in the original.

Not all image formats proposed in the literature strictly adhere to the above definition; the reader is referred to papers describing the original Advanced Forensic Format (AFF) (Garfinkel et al., 2006) and AFF4 container format (Cohen et al., 2009) for a thorough review of evidence container formats currently in use within digital forensics.

As our work builds on the AFF4 work, we recap the key features of this format in comparison to the model described above. The AFF4 evidence container expands on the model by adding address space virtualization and a globally unique referencing scheme. The address space virtualization scheme allows one to construct a virtual bitstream based on mapping sub-bitstreams into a virtual address space. In AFF4 terminology the virtual bitstream is specified by a Map stream. Maps can point to arbitrary sources of bytes: the first such contemplated being an Image stream (a contiguous bit stream composed of compressed or uncompressed blocks of equal size). Such blocks are organized into groups for efficient indexing. The groups are called Segments. The globally unique referencing scheme is used by the Map to refer to objects in the AFF4 system, ranging from byte sequences to entities, and located locally or remotely.

Limitations in current evidence container approaches and related work

The dominant approach to assurance of the integrity of the forensic image is the linear bitstream hash: generally an MD5 or SHA1 hash (or both) of the linear image byte sequence, starting from the first byte of the first storage block, to the last byte of the last storage block. An alternative to the linear bitstream hashes is the use of segment hashes: hashes of equal and fixed size chunks of the source device. In the AFF container format these were typically 1M or 16M, and in dcfldd are configurable.

At high I/O rates, linear bitstream hashes are problematic for scaling acquisition. Such a hash is a relatively expensive operation in terms of CPU resources, with SHA1 currently able to hash at around 600 MB/s on a current generation i7 core (see Table 1) and around 450 MB/s on a current generation i5 core. Unlike segment hashes, linear bitstream hashes cannot be parallelized across multiple cores, accordingly, on low speed CPUs linear bitstream hashes become a bottleneck for high speed acquisition.

Table 1
Single core algorithm throughput for i7-4770 3.4 GHz.

Function	Algorithm	Throughput (single core)
Compression	Snappy	1405.42 MB/s
Compression	LZ4	1538.31 MB/s
Compression	Inflate	39.42 MB/s
Hash	Blake2b	601.87 MB/s
Hash	SHA1	619.23 MB/s
Hash	MD5	745.65 MB/s

We contend that hashes in general are not an efficient use of CPU resources when faced with uniform, sparse data such as sectors filled with zeros. Today's SSD's appear to store such empty blocks as sparse segments and are capable of streaming such sectors at rates exceeding the maximum bandwidth of these standard hashes (current generation PCIe based SSD's are beginning to approach 1 GB/s).

Partial imaging

Inherent in the triage approach is the understanding that the analyst (or tool manufacturer) can make reasoned decisions about the evidential value of data, and that triage should facilitate preservation of the most valuable data while leaving behind the least. The dominant approach to preservation in triage is the Logical Imaging approach. That is, specific file streams and associated filesystem metadata are collected and stored in a logical evidence container. Limitations of this approach include the absence of context (only the metadata chosen by the tool author is collected) and a lack of repeatability (the collected metadata is an interpretation).

An alternative is to relax the completeness requirement of the standard block level forensic image, such that only a subset of the blocks are acquired. Achieving this requires such an image format to be able to record not only collected data blocks, but also which data blocks have not been collected, the “holes”, so to speak. This issue is well recognized in the Volatile Memory acquisition field, due to the prevalence of memory discontinuities in the physical RAM address space, and has led to the adoption of two primary approaches: employ a linear image and fill the discontinuities with a known “innocuous” value (Encase EWF and Garner's DD) or to adopt a section based binary storage format (i.e. ELF, DMP).

X-Ways has recently added support for “Skeleton images” (Watkins et al., 2009) the implementation of which is a raw image backed by a sparse NTFS file. On attachment of an empty skeleton image, subsequently read blocks of an associated device are copied into the skeleton image at the appropriate offsets, such that relevant volume metadata, filesystem metadata, and accessed file content are saved into the image. An MD5 block is calculated and stored for each sector, and saved in a log file.

Non-linear imaging

While ideal for maximizing the read bandwidth of the storage devices such as spinning disks, the general linear approach to imaging prevents the preservation of blocks in any other order.

The "Teleporter" (Watkins et al., 2009) and the AFF4 hash based imaging proposals (Cohen and Schatz, 2010) implicitly proposed a non-linear approach to acquisition, but fell short of addressing the overall integrity hashing approach. While it may seem straightforward to create a linear bitstream hash of the input (albeit out-of-order) bitstream, verifying such a hash is not possible using the AFF4 evidence container in its current form. This is due to there being no mechanism in the AFF4 by which the ordering of all Map segments is recorded.

Compression throughput

Within the field there is a lack of consensus regarding the benefits of employing compression in acquisition. The use of compression allows a tradeoff between CPU and output I/O resources. In I/O bandwidth limited and CPU rich environments, compression can lead to significant gains in throughput and minimization of latency.

The de-facto standard compression scheme in physical images is the use of the Deflate algorithm (Deutsch, 1996) to compress data blocks, as employed by sgzip, EWF, AFF1 and AFF4. The Deflate algorithm is an order of magnitude slower than MD5 and SHA1 – achieving on a single core of an I7 on the order of 40 MB/s. Unlike linear bitstream hashing, such block compression is parallelizable across multiple cores, and as demonstrated by tools such as Guymager, readily achieve speeds in the range of 100 MB/s (Zimmerman, E). In addition to Deflate, AFF v3 supported the use of LZMA, a slower but more space efficient compressor.

Symbolic compression

Byte runs consisting entirely of a single byte value are a common occurrence on storage mediums. These are increasingly of relevance due to the emptying of unallocated space brought about by some implementations of the SSD TRIM feature. While highly compressible, the compression of such data runs using dictionary coding data compressors such as Deflate is inefficient compared with testing for the contents being uniformly of the same byte. Furthermore, storing the compressed version is inefficient in terms of storage consumption.

While the notion of sparse image sections (byte sequences consisting entirely of the byte 0) was contemplated by the AFF4 proposal, it was never formally described. AFF4 provided the *aff4:Zero* stream, which is used to represent data runs consisting wholly of bytes with hex value 0x00, similar to */dev/zero* in Unix. More recently, the EWF2 format has extended this idea by introducing sparse chunks, which enable the specification of chunks containing wholly a particular byte. Such symbolic compression is useful in symbolically compressing erased flash memory, which in that state, contains wholly the byte 0xFF.

Aggregating output throughput

The current model of forensic imaging is generaly based on the paradigm of storing the forensic image on a single storage device. Assuming unlimited CPU and memory bandwidth, this restricts the upper limit of the speed at which imaging proceeds based on the combination of storage device write throughput, storage I/O bus speed, and OS/Filesystem efficiency. For example, when imaging with a Linux based forensic Live CD, one might be able to sustain on the order of 200 MB/s to a commodity SATA drive, however this is might be limited to only 40 MB/s if the only output I/O channel on the suspect computer is USB2.

Assuming sufficiently fast I/O channels, filesystem, and CPU, the primary limiting factor in scaling image throughput is then dictated in current approaches by the write speed of a single hard drive or its bus.

Approaches contemplating spreading image subcomponent writes across multiple destinations are limited in the literature: the original AFF4 work contemplated storing of sub parts of forensic images in arbitrary physical or virtual locations and Farrell (2013), describes a method of copying hard drive blocks across multiple drives simultaneously, utilizing the aggregate I/O throughput, without however, addressing hashing.

The current generation of dedicated imaging devices are beginning to offer "fill and spill" of an image from one storage device to another when full, however this addresses scaling of image size but not rate.

Extending the AFF4 container format for low latency acquisition and analysis

The former section identified a range of limitations inherent in current acquisition approaches. This section describes our proposed solutions to these limitations, by way of additions to the AFF4 container format. While it is not in the scope of this paper to describe the architecture and operation of the software which implements this extended version of the AFF4, we describe some implementation related concerns where relevant.

Symbolic compression

Our implementation performs symbolic compression prior to regular compression, identifying blocks consisting entirely of the same byte. Where such blocks are found, regular compression is omitted. These blocks are represented in the AFF4 Map by reference to virtual streams named after the hex representation of the byte; i.e. "*aff4:SymbolicStream00*" and "*aff4:SymbolicStreamFF*", and so on for the byte range 0x0 to 0xff. We keep as a synonym "*aff4:Zero*" for backwards compatibility.

Partial imaging

Our implementation contains our own independent implementations of the MBR, GPT and LVM volume management schemes, and NTFS, HFS+, FATx & EXTx filesystems. These are used to interpret higher level goals

related to what is to be acquired into concrete regions of blocks of the target storage device.

With the ability to map out the topography of the target storage device in place per the above, supporting partial imaging on top of AFF4 is straightforward. Using the Map abstraction, we define two new virtual stream abstractions: *aff4*:*UnreadableData and aff4*:*UnknownData*. Used similarly to the existing *aff4*:*Zero* stream, the former is used to identify areas where the data is unknown due to read failures. The latter is used to refer to byte sequences within the Map (and consequently image) for which the corresponding data is unknown, due to its not having been read.

On closing a partial image (i.e. an image which has blocks that have never been stored) the all such address runs (holes) are identified in the Map, and a corresponding *aff4*:*UnknownData* region placed in the Map.

Aggregate output throughput

While the original AFF4 work proposed a means of storing portions of a forensic image across multiple volumes, located locally or remotely, it did not explicitly address scaling. We have implemented as an application of the AFF4 what we call striped imaging. Like its RAID namesake, we spread writes across multiple disks, however unlike RAID, we are capable of utilizing the aggregate bandwidth of output channels of differing throughput.

Our striped imaging profile is implemented using the AFF4 primitives in the following way:

1. An AFF4 evidence Container is created on each destination storage device;
2. All such containers have a different and unique Volume ID;
3. In each such container a uniquely identified Image stream is opened;
4. A Map Stream is created in each container such that the ID of the map is identical.
5. For each image chunk, if the chunk is either compressed or un-compressible data, the chunk is written to the Image stream with the smallest number queued writes (measured in bytes and not objects), and the Map streams in each container updated to reflect the storage of the chunk in the stream. If the chunk is a Symbolic Chunk the chunk is stored in the Map streams in each container without writing data to any Image stream.

Non-linear imaging and hashing

The AFF4 container format already supports non-linear imaging (a specific application of which is demonstrated in hash based imaging) by way of the Map abstraction. One aspect of non-linear imaging not considered in the AFF4 work was the means of calculating linear hashes for such images. The current AFF4 design is limited in that it does not record the ordering relationship of blocks stored in the Map. Furthermore, one cannot assume that blocks stored in the Image stream are ordered, due to concurrency in the compression process, and symbolic blocks, stored only in the Map, may be merged, potentially making the original block ordering indeterminate.

Our initial approach to addressing this limitation was to revise the Map serialization of AFF4 to optionally include a record of the ordering of all blocks as received from OS reads of the subject storage device (which we called a *mapPath*). Our experiments identified that such a linear hash of the raw data was a significant bottleneck on low powered CPU's due to a lack of concurrency.

Accordingly, our default approach to hashing is to employ segment based hashing (which we call block based hashing due to the overloaded usage to the term segment in our current vocabulary), and to limit its application to only to non-sparse (symbolic) blocks. This enables hashing to proceed in parallel on multi-core CPU's.

We define a third bevy element (a block hash segment) which stores the block hash of each corresponding chunk of a data stream. The name of the block hash segment is based on the name of the bevy and the type of hash in use. For example, for the compressed block stream named aff4://83a3d6db-85d5/ we might have the following segments as described in Table 2:

We consider hashing of sparse chunks to be a waste of resources and accordingly do not hash those chunks. Symbolic regions, being a component of the Map, are protected by the Map hash. For example, for a Map named *aff4://89313d6db-15d9/* the hash is calculated as

$$\begin{aligned}&aff4{:}mapHash = \\ &sha256(aff4{://}83a3d6db\text{-}85d5/map\ .\\ &aff4{://}83a3d6db\text{-}85d5/index\ .\ \{\\ &aff4{://}83a3d6db\text{-}85d5/mapPath\ \}\ )\end{aligned}$$

$$\begin{aligned}&aff4{:}mapPointHash = \\ &sha256(aff4{://}83a3d6db\text{-}85d5/map)\end{aligned}$$

$$\begin{aligned}&aff4{:}mapIndexHash = \\ &sha256(aff4{://}83a3d6db\text{-}85d5/index)\end{aligned}$$

$$\begin{aligned}&aff4{:}mapPathHash= \\ &sha256(aff4{://}83a3d6db\text{-}\\ &85d5/mapPath)\end{aligned}$$

The period (".") represents concatenation of bytes and the sha256 operations are applied to the content of the named segments. Curly brackets represent optional elements.

A hash of the block hashes of an Image stream is calculated per the below:

$$aff4{:}blockHashesHash = sha256(\ blockHashes_0\ ..\ blockHashes_n\ )$$

The *aff4*:*blockHashesHash* can be calculated as each hash of each chunk is added. In striped acquisitions, there will be multiple Image streams, each of which will have a stream of block hashes, and accordingly, an *aff4*:*blockHashesHash* over those. The ordering of those in the blockHash calculation above is determined by the ordinal of the corresponding Image stream in the Map index.

Finally, recognizing the importance of having a single hash to record and communicate which applies to the entire image, we generate a single SHA256 hash called

Table 2
Bevy segments related to segment hash.

AFF4 name	Segment type	Purpose
aff4://83a3d6db-85d5/00000032	Data segment	Chunks of stored data
aff4://83a3d6db-85d5/00000032/index	Chunk index segment	Offsets within data segment for chunks
aff4://83a3d6db-85d5/00000032/blockHashes.sha1	SHA1 block hash segment	SHA1 hashes of chunks within data segment.

aff4:*blockHash* based on the concatenation of the hashes of the block hashes of each stream and the associated Map hashes as follows

$$aff4{:}blockHash = sha256(blockHashesHash_0 .. blockHashesHash_n . mapPointHash . mapIndexHash . \{ mapPathHash \})$$

The above hashing approach enables the blockHash to be calculated without any re-reading of data from the image.

Compression throughput

As identified in the prior section, non-symbolic compression is the single most costly compute related operation involved in contemporary physical acquisition. While parallelizing the operation has succeeded in achieving rates in the realm of 100–200 MB/s, imaging remains, with high I/O rate source devices, CPU bound. Scaling requires more cores or faster algorithms. Our approach is to employ both.

We modify the AFF4 container schema to support configurable compression schemes and apply lighter-weight compression algorithms: the Snappy algorithm developed by Google and the LZ4 algorithm used in the ZFS filesystem. In our benchmarks both algorithms exhibit single threaded compression at around 1500 MB/s on a single core of a dual core I7 (around 30× faster than Deflate).

In order to support such configurable compression, we add a property called "*aff4*:*compressionMethod*" to the AFF4 Stream object, with the corresponding values being https://code.google.com/p/lz4/ and http://code.google.com/p/snappy/ used to specify the algorithm used for that stream. The following example demonstrates the Snappy compression method specified for the stream *aff4://0466b8fb-9af0-4ef2-b36c-8b0d90fc0ac2*.

```
<aff4://0466b8fb-9af0-4ef2-b36c-
8b0d90fc0ac2>
      a  aff4:stream ;
      aff4:CompressionMethod
<http://code.google.com/p/snappy/> ;
      aff4:chunk_size
"32768"^^xsd:int ;
      aff4:size
"294912"^^xsd:long ;
```

While these algorithms do not compress data as well as the Deflate algorithm, they do allow a CPU/bandwidth tradeoff which assists in achieving higher effective speeds. In situations where the available CPU exceeds output bandwidth (for example cross-internet), usage of Deflate or even LZMA may be warranted.

Experimental methodology

In order to demonstrate the limitations identified in existing evidence container approaches and to demonstrate proof of concept, a testbed focusing on high bitrate acquisition was built. Table 3 summarises the hardware used in the testbed.

Sample preparation

The target device (test drive) was populated with data as follows:

1. Windows 8.1 installed from empty to the whole drive (around 10.2G allocated)
2. High entropy data added to filesystem from Linux /dev/random (38.4 GB)
3. Windows main filesystem shrunk by around 100 GB
4. A new volume and NTFS filesystem created in the free space.
5. From the GovDocs1 (Garfinkel et al., 2009) corpus, the contents of Zip files 001-075 were extracted into the fresh volume, then the files corresponding to 001-040 copied such that a second set of those files exist on the filesystem (59.8 GB).

Compression characteristics

The drive was linearly acquired using our system using the Snappy compression algorithm, and a graph generated from the image such that, the y-axis corresponds to the chunk size (in bytes) and the x-axis corresponds to ordinal of the chunk (see Fig. 1). The uppermost horizontal line, (red points in web version) indicate the size of chunks pre symbolic and snappy compression, and the green points (in web version) the size of chunks as stored, post symbolic and Snappy compression.

With reference to Fig. 1, the leftmost vertical grouping labeled "A" corresponds to around 10.2G of files related to the windows installation, and shows Snappy compression has yielded significant gains. Similarly, the GovDocs subset (the rightmost vertical grouping labeled "E", roughly corresponding to 59.8 GB) shows significant stored size reductions due to Snappy compression.

Table 3
Testbed hardware.

Element	Details
Target device	240G Intel 730 SSD
Computer 1	4 core i7-4770R 3.20 GHz CPU system
Computer 2	2 core i5-3337U 1.80 GHz CPU system
USB3 Bridge(s)	Orico USB3/eSATA bridge
Evidence HDD(s)	Toshiba 2 TB 7200RPM Commodity SATA

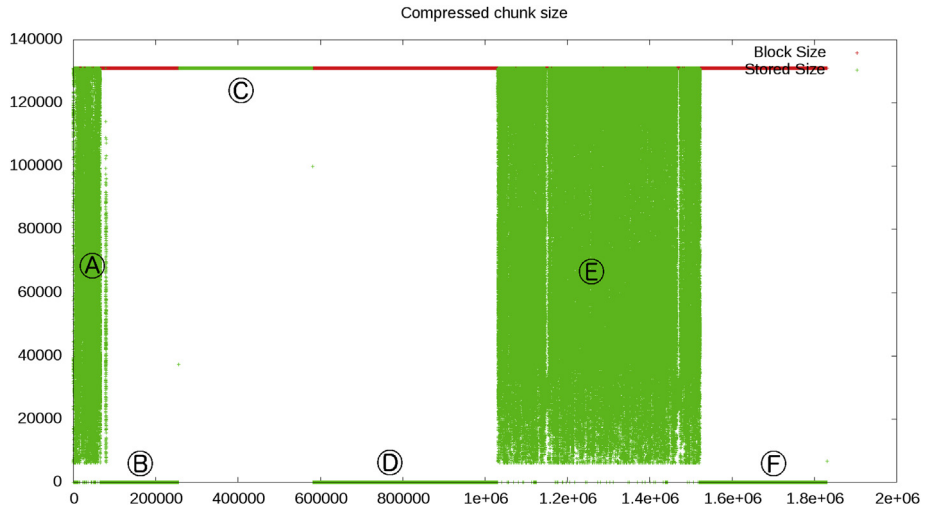

Fig. 1. Stored chunk size vs chunk address for sample image.

The uppermost horizontal grouping labeled "C" corresponds to the 38.4 GB of high entropy (random) data in the filesystem, and is uncompressed. Much of the lowermost horizontal groupings labeled "B", "D", and "F" correspond to sparse (unused) portions of the drive that have been symbolically compressed.

Compression effectiveness

Two separate test runs generated a linear image of the sample drive connected via SATA3 to Computer 1, and a single hard drive via USB3 as the image destination. The variable in the experiment was the usage of compression (Symbolic Compression remains active in both runs). Samples of the read I/O rate of the sample drive were collected on a 1s basis using the Linux iostat utility.

Fig. 2 is a graph of the results, where the read I/O rate of the target drive is graphed vs elapsed time, the results for compression enabled are graphed in red points (in web version) and the results for compression disabled are graphed in green points (in web version).

With reference to the figure, when read from the left, it is apparent that the rough shape of both plots is in general similar in regard to both sparse (empty) blocks (labeled "A", "B", and "C") and the high entropy (random) data (labeled "D"). In the former case, acquisition proceeds at close to the wire speed of the SSD, which, according to Intel, is around 550 MB/s (Intel, 2014). This is due to the negligible load on both the CPU and output I/O channel due the effect of symbolic compression.

In the case of high entropy data, the I/O rate hovers at around 200 MB/s, with the maximum write throughput of the destination evidence hard disk being the bottleneck.

The primary variance in the two plots is the sections corresponding to the Windows OS (labeled "E") and the GovDocs files (labeled "F"): with compression the I/O rate is higher, while without compression, the I/O rate tends towards the 200 MB/s limit imposed by the output I/O channel.

Acquisition completes more quickly using compression than without, with acquisition completing 2:08 faster with Snappy compression.

Striping effectiveness

The effectiveness of striping acquisition across multiple output channels was tested by employing the same hardware setup as the prior experiment. Three test runs were conducted involving the generation of linear images, with the variable being the number of output channels (USB3 connected disks).

The results of the runs are graphed as Fig. 3, with the read I/O bandwidth graphed in red (in web version) for the run using only a single destination drive, and in green (in web version) for the run using two destination drives, and in blue (in web version) for the run employing three destination drives.

With reference to the figure, the high entropy data section contains the most variability, being I/O bound on output for the run employing a single destination drive, sitting on around 200 MB/s (label "A"). The addition of a second destination drive significantly increases the effective bandwidth for this section, averaging at around 400 MB/s (label "B"). The addition of a third destination drive increases the effective bandwidth again, to approaching 500 MB/s (label "C").

Table 4 summarises the total acquisition time for the three runs.

The impact of hashing

In order to validate our theory that stream based hashing is a bottleneck for CPU constrained environments, the following test was conducted. Computer 2 (a dual core i5) was paired with two destination drives (an aggregate of 400 MB/s). As before, we acquire a linear image, and in this test, vary the hashing scheme between stream based hashing and block based hashing. Stream based hashing

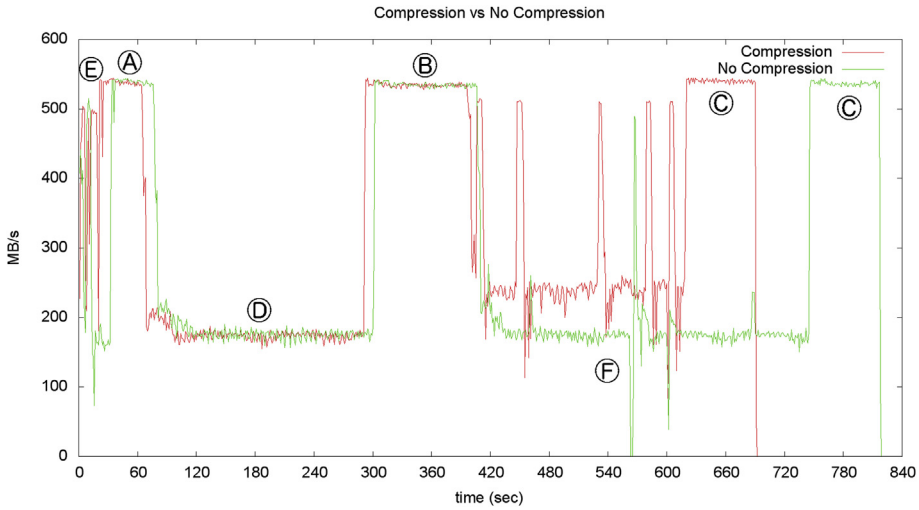

Fig. 2. Linear acquisition throughput is increased by compression.

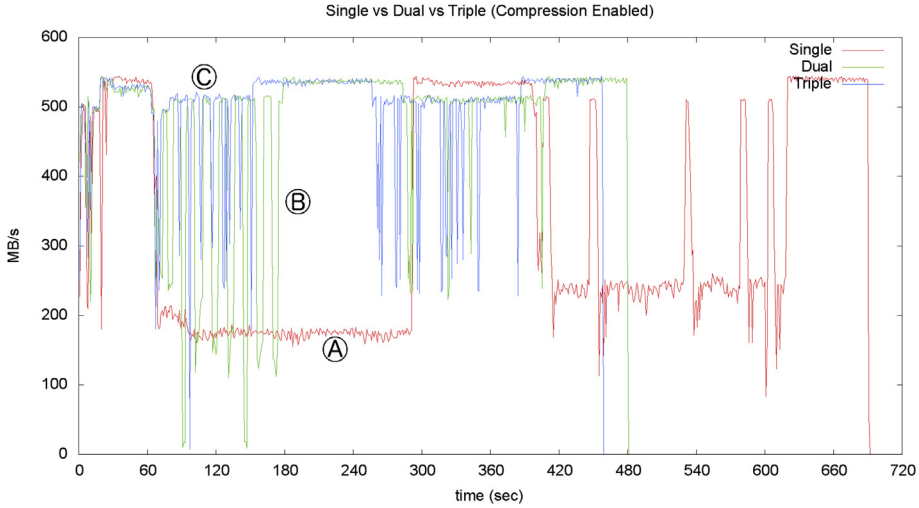

Fig. 3. Aggregated I/O channels increase linear acquisition throughput.

takes a hash of ALL data ($LBA_{start} \rightarrow LBA_{end}$) including blocks that are symbolically compressed, whereas block based hashing is as described in Section 5.4, omitting symbolic blocks.

Fig. 4 plots the read I/O bandwidth of both tests. With reference to the figure, it is apparent that for the sparse section of the disk the read I/O rate sits at around 350 MB/s (the green plot (in web version), label "A"), effectively constrained by the single core performance of SHA1 hashing for this CPU (which we have separately confirmed by a separate test). For the same section of data (labeled "B") under block based hashing, the bottleneck is, as with all prior tests, approaching the maximum read I/O bandwidth of the combination of the target device and SATA3 bus (around 530 MB/s).

A similar test employing Computer 1 (the 4 core i7) showed no effective difference between the usage of stream and block based hashing, most likely due to the single thread hashing performance exceeding the maximum input rate.

Table 4
Total acquisition time by number of destinations.

Number of destinations	Total acquisition time
Single destination	11:29
Two destination	8:00
Three destination	7:40

Effectiveness of partial acquisition

In order to evaluate the effectiveness of partial non-linear acquisition, test runs were undertaken involving an allocated-only acquisition of the sample using Computer 1, with two output channels.

The graph of read I/O rates of the test, along with the results of a linear acquisition for the same setup is presented as Fig. 5.

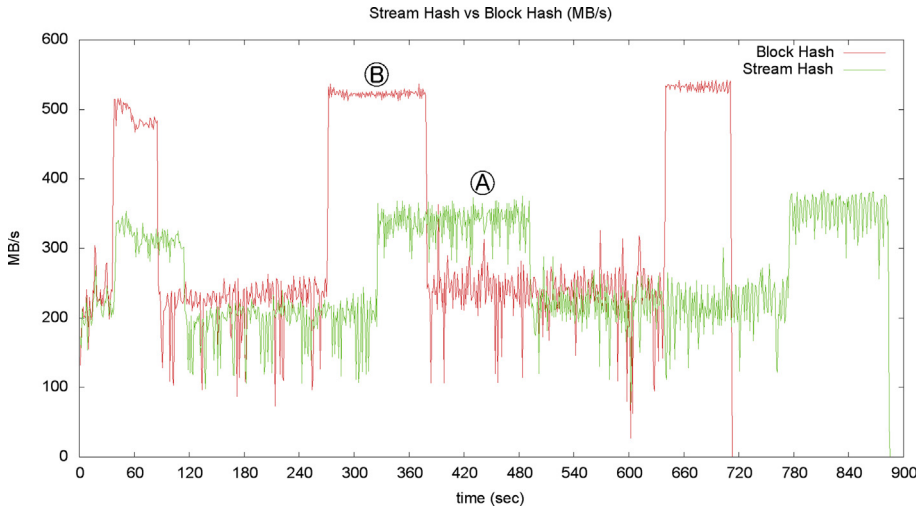

Fig. 4. Block based hashing gives higher throughput than stream based hashing in linear acquisition.

With reference to the figure, it can be seen that an allocated only acquisition (115 GB in total) completes in just over half the time that it takes for a full linear acquisition at an average rate of around 400 MB/s.

Summary of total acquisition times

Table 5 summarizes the total acquisition times associated with varying both the number of stripes and the scope of acquisition (linear full vs. non-linear allocated) for Computer 1 and the test drive. With reference to the table, for linear acquisition of the whole disk, the average acquisition rate reaches what we have observed to be the maximum read rate of the SSD in our test system when using 3 stripes as output (a 35% increase in throughput over one stripe). Using two stripes yields a round a 30% improvement (Table 6).

The single stripe allocated acquisition has a comparatively lower acquisition rate, due to the removal of the easily compressible and low cost storage sparse regions. Adding a second stripe to an allocated acquisition nearly doubles the throughput, consistent with being output I/O bound. Adding a third stripe, while giving an appreciable gain of 40 MB/s, is less than the gain of adding a third stripe for linear acquisitions. We speculate that this may be due to the more random nature of this workload impacting our scheduler or the SSD's read-ahead.

Comparison with existing approaches

In order to compare the effectiveness of the approach in comparison with existing common approaches, we undertook 6 further test runs, using the test sample and Computers 1 & 2 and a single image destination. The tests

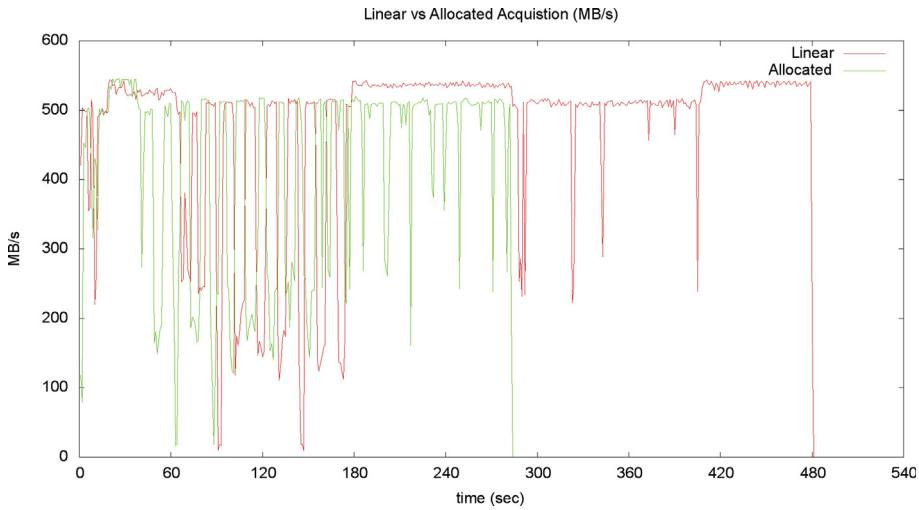

Fig. 5. Non-linear partial acquisition reduces acquisition time.

Table 5
Total acquisition times for proposed approach.

Number of destinations	Total time (s)	Average rate (MB/s)
Wirespeed, 3 stripes, linear	7:30	533
Wirespeed, 2 stripes, linear	8:00	500
Wirespeed, single evidence disk, linear	11:29	348
Wirespeed, 3 stripes, allocated	4:17	447
Wirespeed, 2 stripes, allocated	4:42	408
Wirespeed, single evidence disk, allocated	8:21	229

recorded the total acquisition time using FTK Imager and X-Ways running from a WinFE forensic live CD, based on Windows 8.1. No graphs were generated due to an absence of performance monitoring infrastructure on the LiveCD.

FTK Imager 3.1.3.2 was executed using compression setting 1 (fast compression) and using standard stream hashes. There is no setting to configure the hash(es) in use, so the defaults (SHA1 & MD5) were used. X-Ways 18.0SR-8 was executed using fast adaptive compression, using the default number of compression threads (2*CPU core count), and only SHA-1 hashing. Both applications use Deflate based compression. The results are summarized in Table 5.

With reference to the table it is apparent that for XWays Forensics, the cost of the Deflate algorithm is largely negated by the usage of threaded compression and an abundance of CPU resources for Computer 1, whereas with low CPU resources (Computer 2), the usage of the Deflate algorithm lowers throughput by more than half.

Recalling that the maximum write rate of the destination drive has been measured to be around 200 MB/s, at that rate, writing 240 GB of raw data should take around only 20 min. Comparing the measured acquisition times with this, it is apparent that for FTK Imager, even with an abundance of CPU resources, the Compression/Output throughput tradeoff of using compression is inconclusive. With limited CPU resources it is detrimental. For X-Ways, under abundant CPU, the usage of compression yields significant gains, but, like FTK imager, is detrimental to throughput on lower power hardware. The proposed technique yields significant throughput gains in both situations.

Limitations

The above experiments do not include an image verification phase, which is generally used in practice.

Table 6
Total linear acquisition time in seconds.

Acquisition application	Computer 1	Computer 2
FTK Imager	20:10 (198 MB/s)	37:38 (106 MB/s)
X-Ways Forensics	13:58 (286 MB/s)	33:23 (120 MB/s)
Proposed approach	11:29 (348 MB/s)	15:08 (264 MB/s)

Verification under our proposed extensions to the AFF4 format enjoys commensurate throughput increases due to faster compression algorithms and the absence of hashing symbolic chunks.

Conclusions

In this work we identified the limitations inherent in current forensic physical acquisition approaches which limit both the rate at which acquisition may proceed and the extent to which targeted subsets of physical disks can be acquired. The contributions of the paper are the proposal of a number of extensions to the AFF4 image container format which allow acquisition to proceed at high bitrates through increased concurrency, lightweight compression schemes & horizontal scaling of image storage devices. These proposed extensions have been systematically evaluated through experiments which quantify the limitations of former approaches under a variety of factors adversely affecting acquisition throughput, and proof of concept demonstrated by acquiring images at target bitrates of around 500 MB/s.

Finally, it has been demonstrated that partial non-linear images may be taken at bit rates that exceed current physical acquisition approaches.

Acknowledgments

The author thanks the reviewers for their detailed and considered feedback.

References

Bertasi P, Zago N. FASTDD: an open source forensic imaging tool. In: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on. IEEE; 2013, September. p. 485–92.

Cohen M, Garfinkel S, Schatz B. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. Digit Investig 2009;6:S57–68.

Cohen M, Schatz B. Hash based disk imaging using AFF4. Digit Investig 2010;7:S121–8.

Deutsch P. DEFLATE compressed data format specification. version 1.3, rfc1951, IETF. 1996.

Farrell, (2013) Digital memory imaging system and method, Patent GB2503600.

Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. DFRWS 2009, Montreal, Canada. 2009.

Garfinkel S, Malan D, Dubec KA, Stevens C, Pham C. Advanced forensic format: an open extensible format for disk imaging. In: Advances in digital forensics II. New York: Springer; 2006. p. 13–27.

Intel. Intel Solid State Drive 730 Series. 2014 (accessed 26.09.14), http://www.intel.com.au/content/www/au/en/solid-state-drives/solid-state-drives-730-series.html.

Rosen A. ASR expert witness compression format specification. 2002. Accessed Sept 2014, http://www.asrdata.com/SMART/whitepaper.html.

Roussev V, Quates C, Martell R. Real-time digital forensics and triage. Digit Investig 2013;10(2):158–67.

Tableau. Tableau TD2u forensic duplicator. 2014 (accessed 26.09.14), https://www.guidancesoftware.com/products/Pages/tableau/products/forensic-duplicators/td2u.aspx.

Watkins K, McWhorte M, Long J, Hill B. Teleporter: an analytically and forensically sound duplicate transfer system. Digit Investig 2009;6:S43–7.

Zimmerman E (accessed 26.09.14), http://goo.gl/2h9pCd; 2013.