Incentivised Distributed Computation for Registered Digital Content Search

TECHNICAL WHITEPAPER

Samuel Brooks
Veredictum

July 2017

Abstract

We explore the feasibility of using a distributed computation system to download and process video and audio files found on the open internet to detect watermarks. This, in effect, allows the system to "search" for any content that has been registered with a watermark, including pirated content, rather than using traditional file-based search methods. We present a centralised coordination function for the distributed compute network, a blockchain-based cryptocurrency incentivisation scheme and a dispute resolution mechanism to thwart bad actors.

v0.9.1

A REFERENCE IMPLEMENTATION OF THIS DESIGN MAY CHANGE FROM THE DESCRIPTION CONTAINED WITHIN THIS DOCUMENT

Contents

1 Introduction

This paper discusses a novel method for searching for watermarked video content on the internet.

Tracking the movement of digital content online presents significant challenges due to the client-server architecture of the internet; files are hosted on opaque remote servers and one must browse, download, and inspect the contents of the file in order to understand if a certain piece of digital content is present. Further, digital files are easily and frequently re-distributed to uncontrolled and unauthorised web servers (digital piracy). File names can be easily changed, files can be compressed, down-sampled, and otherwise transformed, all the while preserving the content to be suitable for human viewing.

Given this client-server architecture, telemetry from digital files (status information sent from a remotely configured device back to a central location) is practically impossible. File telemetry would require the cooperation from the remote server to install additional software and transmit information relating to its digital content inventory. Server owners have no particular incentive to distribute an accurate catalogue of the digital content held on their servers.

Blockchain-based cryptocurrencies can incentivize large networks of machines to perform computations as proofs-of-work [1]. Within the context of video content being commonly pirated (such as freebooting between YouTube and Facebook), this paper investigates the technical feasibility of an incentivised, distributed search capability for digital creative content that has been "marked" using modern steganography. This capability involves the downloading and processing sectors of the internet via a distributed network of computers (rather than a cluster of dedicated centralised servers), in order to develop a profile of where pirated content has been distributed across the internet. We focus our attention on video content in particular (being a superclass of audio content).

2 Overview

2.1 Problem statement

Video piracy is a significant parasitic cost to major film studios and small content creators alike. Several studies have suggested that the cost to the US economy alone is up to $20B USD [2].

In 2016, Facebook generated over $30 billion in revenue, and yet 73% of the top-performing videos distributed on the social network were stolen from content creators who had originally uploaded their work to YouTube [3]. This is more commonly known as "Freebooting".

Currently, in order for a content creator's stolen content to be removed from Facebook, they must:

1. initially be alerted to the fact the content has been stolen and re-uploaded onto Facebook, and
2. complete a lengthy online form as well as upload the original content to Facebook in order to help prove the copyright claim and have the material removed.

While the YouTube-Facebook interplay is a high-profile example of the nature of the problem (a recent YouTube video discussing the subject ironically had more views on Facebook [4]), it is not the only place where it occurs [5]. A solution that could eliminate this kind of inefficiency would have a significant reduction on the revenue drag for online video, resulting in both higher-quality content and lower prices for consumers.

2.2 Centralised Versus Distributed Computation

A single entity intending to search the open web for a single piece of content is facing the manual review of an intractable number of files. In the case of Facebook, approximately 1 Petabyte of data is added to their servers each day, around 1/3 of this as video [6]. In order to independently verify the copyrights of uploaded video, this 300TB of new video data would need to be downloaded and examined on a daily basis. Using a centralised cloud service such as Amazon Web Services (AWS), this analysis would cost of the order of $100,000 USD [7].

2.3 Solution summary

This paper explores the technical feasibility of a content download and review function performed by a distributed network of incentivised nodes. The system relies upon the steganographic pre-processing of digital files (watermarking) and the registration of ownership to a public blockchain. Then consuming the search space suspected of serving copyrighted

material would provide a profile of when and where the content has been re-distributed (illegally and legally).

Our system is defined in two key phases:

1. **Content Encoding and Registration:** In the first phase, content is encoded with a unique identifier using a centralised server. We then register this identifier to a public blockchain (e.g. Ethereum) and distribute the content to its intended destination.
2. **Content Search and Reporting:** We then assume that some piracy event will occur with some significant probability. We then search, download and review a number of suspect video files using a set of incentivised computers, and report results back to the subscribers of the service. Coordination of this phase is centralised and the computational effort is distributed.

We make some assumptions regarding the cost of latent processing and bandwidth resources available in devices distributed across the internet and leverage this to propose a practical content search engine for registered content. We describe an incentive scheme, inspired by Ethereum, designed to ensure that computations are verified to within a high probability.

With a more mature distributed technology ecosystem, it may be possible to ultimately perform the centralised tasks in a distributed fashion. We leave this possibility to future work.

3 High Level Solution

3.1 Definitions

3.1.1 Digital content and files

We define digital *content* as distinct from a digital *file*. Content is defined as static and files are defined as polymorphic; the file is simply the container for the content. The properties of a digital file can change significantly but the nature of the content remains the same (e.g. a music file can be down-sampled and have its filename and file type changed but the file can still contain content of Chuck Berry singing Johnny B. Goode). The *content* is the valuable asset we wish to protect; the file itself has no intrinsic value.

3.1.2 Client and server

We refer to a *client* as being the application that a node will download from a central server to participate in the system. We refer to this centralised, orchestrating server as the *Coordinator*.

3.1.3 Logical actor definitions

3.1.3.1 Subscribers

Subscribers are customers who pay for the search capability. It is envisioned that the fees associated with the functionality are paid on a subscription basis so that the expected pay-through to miners is more predictable.

3.1.3.2 Miners

A miner is a person participating in the distributed computation network to earn cryptocurrency. For the purposes of this whitepaper we assume a single miner per client, however in practice there may be multiple clients under the control of a single person or company.

3.1.3.3 Clients and Nodes

A client is a single instance of the software on a miner's computer. We define a node as a single virtual machine (VM) within a client. There can be multiple VMs instantiated per client if the underlying hardware is powerful enough.

3.2 Solution Phases

3.2.1 Phase 1: Content Encoding and Registration

The process of content registration is outlined below. These tasks are assumed to be operational as this paper mainly deals with the distributed search engine, however, resilience testing results from our in-house steganography software is also covered:

- Securely establish the identity of the content owner who wishes to register a new piece of creative content.
- Generate a new, unique identifier for the content.
- Steganographically embed this new content identifier within the file.
- Register the content identifier to the content owner using a smart contract on a public blockchain.
- Distribute the file to its approved destination(s).
- This process is ongoing.

3.2.2 Phase 2: Content Search and Reporting

The search phase consists of:

- Crawling servers of interest to compile a list of video files available for download.
- Sorting the search space to attempt to prioritise files that have a higher probability of containing an identifier.
- Partitioning the search space into a number of sectors such that each sector is able to be downloaded and processed by a single node in some reasonable time period (e.g. a sector size of 1GB).
- Recruiting a set of computers to download and process each sector.
- Returning the results to a central server (the Coordinator) to create a profile of files that contain content identifiers for that cycle.
- This process then repeats once per cycle time.

3.3 Video steganography

Video steganography is the practice of concealing a message within a video. Modern techniques exist that enable a video file to be transcoded into a new file that contains an embedded message that is both invisible and inaudible to regular viewing. This provides a method of 'watermarking' the content so that when it is later examined, a unique content identifier can be recovered that associates the content with the identity laying claim to the copyright. Given the immutability of the blockchain and the strength of modern cryptography, this claim becomes extremely strong.

In order for the marked video to remain valuable, it must be of near identical video quality as well as be resistant to transformations designed to defeat the reproduction of the embedded identifier (such as horizontal flipping of the video or re-recording).

Modern steganographic techniques achieve this with no noticeable degradation in audio-visual quality and with high transformation resistance (resistance strength is exponentially related to the quality of the resultant file). This means that even with a high degree of transformation, recovery of the embedded information may still fail to detect the watermark, but only where the video quality is so degraded that the content becomes effectively worthless for consumer applications.

The results of robustness testing of our steganography software are provided below.

3.4 Search space preparation

A digital content search space is indexed to produce a list of target URLs for further processing. This process initially utilises a web crawler to produce a raw index of video files from a list of target domains. The crawler operates according to certain conditions optimised for our application, such as: do not follow hyperlinks and only index seed domains.

The search space is sensibly bounded as to maximise the probability of finding marked digital content. This can be done for example through prioritising domains and subdomains that are suspected of holding unauthorised content. Bayesian probability and learning algorithms can be applied to both (a) minimise the search space over time in order to reduce the compute resources required and (b) rank individual files by their probability of being watermarked. This information can be aggregated to form a 'piracy reputation' for web servers and specific sub-domains, such as individual users on social media.

Once the search space has been sorted, it is segmented and provisioned to the connected nodes in the distributed computation network.

3.5 Client download, configuration and connection

Participating users in the distributed computation network download and install an application (the 'client') from a central server (the 'Coordinator'). A precise copy of the correct version of software must be installed due to the need to take snapshots of the internal state of the application as it performs computations (see below sections relating to verifiable computations). This can be verified via regular checksum matching.

The client contains an internal virtual machine cluster and manages the interfaces to both the Coordinator and Arbitrator. The client also manages the file download and state hashing of the virtual machine and provides a user interface.

Account log-in by the miner is required via the client. This ensures the miner's reputation can be tracked and that sufficient cryptocurrency is available in their account to form a deposit. These are necessary for participation in the network in order to enforce the design of the game-theoretic computation reperformance. The process of acquiring cryptocurrency is beyond the scope of this document.

The client is also intended to manage scaling; if a miner has access to greater computer resources (such as a server farm), the client will check the user-defined settings and then automatically spawn additional nodes to use the available resources. This effectively homogenises the nodes in the network, which simplifies the matching of nodes by the Coordinator and enables a greater number of connections to be made for better randomness (higher value of n for n choose k).

Once the client has made a server connection, it submits information relating to the user-defined settings and details of the available hardware resources on the device.

Once a cut-off period has been reached, the Coordinator:

1. Stops accepting new nodes for the next search cycle, but starts a queue for the one after. A search "cycle" is one period where a finite set of file URLs are collated for download and processing, and one tranche of a finite amount of cryptocurrency is gathered from subscribers to pass through to the distributed computation network. The Coordinator also analyses the total computational capacity in the network and assigns the sub-networks ("subnets"): nodes are grouped into small sets (e.g. 2 or 3) that have been chosen to perform the computation on the same search space partition. Nodes are chosen in such a way as to minimise the possibility of collusion, such as random selection with geographical segregation. The subnets are assigned an ordinal at random, in the special case that the number of subnets exceeds the number of subspaces, the remaining subnets are re-assigned their ordinal at the start of the next cycle. We do not consider if this occurs past more than one cycle.
2. Provisions nodes with their allocated URL list (subspace). Note that by this step the number of subnets equals the number of subspaces.

Note also that there is no requirement for nodes allocated in the same subnet (i.e. peers) to connect to one another directly; the Coordinator does not provide any information relating to what nodes are in the same group. Note too that the subnet size can decrease to below 2; a subnet size of 1.5 would indicate that every second node's state is replicated at random, meaning that each node would know that, on average, half of their submissions are checked by an independent node, but would not be aware of which submissions specifically. This is explored further in the discussion of reperformance design.

3.6 Client processing

All nodes that are connected in the distributed network utilise their bandwidth and processing power to download and process their allocated sector of the total search space.

In order to verify that nodes are performing the computation correctly and not returning bogus results, at regular intervals within the virtual machine, the state is hashed to form a Merkle tree. Merkle tree roots are then submitted from the network back to a centralised module for comparison. If all the Merkle roots are identical for all nodes in the subnet, this means that either all nodes have performed the computation correctly, or they have perfectly colluded in an attempt to fool the system. Attack surfaces and verification of computations are explored later in this paper.

Once the allocated search space has been processed, the recovered content identifiers are also submitted back to the central Coordinator and compared. If the results are consistent, then no more action is taken by the client until the next search space is provisioned; the results are deemed valid.

3.7 Content telemetry

After the distributed network has resolved the content identifiers from the network, the submissions are collected by the Coordinator module into a central database. We now have an up-to-date profile of what content exists across a number of web servers and this can be queried by subscribers (with appropriate access permissions). The information can be used to automatically send offending servers a Digital Millennium Copyright Act (DMCA) Take-Down Notice or serve an invoice or Letter of Demand. The amount requested can be manually configured or automatically calculated based on simple metadata metrics, such as how many views the video has received.

In summary, network orchestration manages the following process:

- Miner downloads the client and is authenticated via username and password.
- Client checks to see if the system is running any other clients.
- Submission of deposits where necessary.
- Client collects system information, internet connection diagnostics and user settings, and then submits this information to the central Coordinator.
- The Coordinator creates a network pool and a sorted search space, then allocates the URL sets to the nodes.
- Client downloads and processes the URLs and returns state information (Merkle tree).
- Coordinator collects state results from the network's computations before finally collecting the watermark results to form a searchable database of URLs for registered content.

3.8 Client compensation

The value transfer between the subscriber (the customer wishing to search for their registered content) and the miner (the service provider performing the download and decoding) is facilitated via a cryptocurrency token. This payment for the expenditure of computational resources is managed via a smart contract. Fees for the task are deposited, in tokens, by the Coordinator into the contract for the benefit of each node in the network before the computations have been initiated. Upon the successful completion of the submission of their states, the Coordinator sends a transaction to initiate the update of account balances based on how many jobs were completed (how much computational effort was expended). Initially, payment is made into an escrow account to be held for some time period to ensure any bogus results can be challenged. If the time elapses and no challenge over the results has occurred, the tokens are unlocked and the node can retrieve them. At this point, the person who earned the cryptocurrency is able to then sell it on an exchange back to subscribers.

The smart contract is also required to hold the deposits of all participating nodes. Deposits must be made before each node is issued with its partition of the search space. The deposits are required to ensure that a penalty can be applied in the event that a node submits incorrect information (i.e. when the states do not match, or when a challenge is initiated).

The cross-checking of state submissions is performed centrally. Ideally, this calculation would be performed by public blockchain infrastructure, however our system generates far too much data for this to be feasible with the current state of technology. Hence, this “trust” must be placed within the Coordinator in performing the function of verifying computations of remote nodes correctly. This is important because it is the checking of the state submissions that determines whether a miner has acted properly; only on the basis that all the states agree can the payment to the miner be approved.

3.9 System Components

Our system is logically segmented into the following high-level software components:

- **Web Application** (web application for users to interact with the system)
- **Transcoder** (application to embed content IDs into video)
- **Registrar** (smart contract registering ownership of content)
- **Indexer** (search space web crawler)
- **Sorter** (search space sorting algorithm)
- **Coordinator** (central application coordinating the distributed nodes)
- **Client** (the application running on the node's machine)
- **Arbitrator** (central coordination of state matching and resolution)

These will be explored individually in the sections below.

4 Assumptions and Parameters

4.1 System parameters

The following parameters are defined. These will be used for descriptions within this whitepaper and any future technical addendums:

- Cycle time period, T_{CYCLE}
- Set of all participating nodes per cycle time, N
- Cross-validation parameter (number of nodes in subnet), v
- Node number, n_x
- Set of all files of interest on the internet, F_i
- Set of all files within the search space per cycle time, F (subset of F_i)
- Set of files per cross-validating network "sub-space" per cycle time, f (subset of F)
- Total size of all files in search space per cycle time, S
- Total size of all files in a sub-space per cycle time, s_i (subset of S)
- Proportion of files consumed that have a watermark, P_w
- Average bandwidth of a virtual node in the network, C_{BAND}
- Average measure of processing capacity of a virtual node in the network, C_{CPU}

4.2 Assumptions

- The cost of central processing is greater than the cost of distributed processing by a significant factor; while $/CPU cycle might be cheaper in a centralised, vertically integrated solution, distributed computers will be able to utilise idle hardware, thus purchasing centralised computation is not required (see Appendix B).
- It is possible to strike an appropriate balance of parameters in the reference implementation to ensure a viable enterprise, including costs, incentives, profits and customer pricing. Note that if we assume a fixed search space, the price per customer decreases with additional customers. Power consumption for the client and the cost of electricity are a part of this cost profile and are explored separately.
- The system makes economic sense such that sufficient content piracy occurs of a type that our system is able to identify.
- Participating nodes are rational actors that seek to maximise their own profit or minimise their losses. We assume that sufficiently large economic penalties are effective to deter the submission of bogus results (e.g. the loss of a deposit equivalent to 10 times the available award for the computation). This would be applied to nodes found to be attempting to claim awards without expending resources (cheating).
- The method for defining, encoding and decoding a content identifier into a video file is relatively efficient when compared to alternative methods (e.g. frequency-domain video waveform reduction to generate a content fingerprint).

- The proportion of data available on servers to download, F, to the ability to download and process these files (i.e. bandwidth and CPU resources) remains constant over time given the general increase in both over time due to technological advancement.
- Sufficient computational resources can be recruited in order to search a meaningful number of files (i.e. there are more than a trivial number of participating nodes).
- Latent network computation capacity will vary by geography and demography. We assume a single average of these parameters (bandwidth and compute) for the purposes of this whitepaper.
- The number of files available to download is typically greater than the number of nodes available for processing those files: $F > N$.
- Assuming a live system, we search only for newly pirated content. There is of course the potential to search for legacy files however this is beyond the scope of this whitepaper.
- File metadata is not 'tumbled' (made dynamic over time) such that the indexer is fooled into thinking an existing file has been newly uploaded given new parameters, such as file name. This may also form a specific attack vector by troublesome web servers.
- It remains feasible to perform searches for new videos that are made available from Facebook and other platforms, and target servers do not take significant measures to block communications from the application.

5 Transcoder

5.1 Watermarking

Our distributed content search engine is based on having a reliable and cost-effective steganographic transcoder that is used to embed content identifiers within a video or audio file with negligible loss of quality and high transformation resistance.

We have tested such a tool in preparation of this whitepaper. We start by providing the encoding application with a short binary message to encode into the given file. Details of how this occurs are not provided due to their commercial sensitivity. We then attempt to destroy the retrievability of the message through various file transformations. Below is a brief summary of results which proves a reasonable level of transformation resistance:

File type: MP4
File length: 120 seconds
Message length: 8 bits

TYPE	DESC	RESULT
Rerecording	Video was played on a monitor and recorded on a hand-held smartphone. The phone re-encoded the file to a new bitrate and framerate, as well as recorded ambient environment audio noise.	Pass: message recovered.
Small file length	File was segmented into 20-second chunks, and the decoder was run over each to attempt to retrieve the message. Longer messages will require longer minimum file lengths.	Pass: message recovered.
Frame within a frame	Video image was cropped by 20% on all edges.	Pass: message recovered.
Reencoding	File is re-encoded into a new file format (from .MP4 to .AVI)	Pass: message recovered.

These results demonstrate the feasibility of this technology and forms the basis of our content search engine. In the future, such encoding may be able to be performed quickly and efficiently in hardware at the time of recording with all new files generating a native embedded identifier that can be quickly registered. For now, we assume a new file enters the ecosystem via a manual, paid process.

5.2 Frequency signatures

Modern video is encoded in compressed form through frequency domain superposition [8]; raw video in the form of high-resolution, high-framerate bitmaps simply generates an unmanageable amount of data. Even modern video cameras transform input signals into the frequency-domain on-chip, before the video file is saved to the device.

Our current watermarking technology may be able to be optimised through the use of a low-pass filtering technique for video files. This would effectively create an unwatchable version of the video for a human, but still appears unique to a machine. In this way we can create a frequency spectrum signature for the video by only persisting low-frequency information (retain too much high-frequency information and the video becomes susceptible to transformations designed to defeat our anti-piracy measures). This may be similar in methodology to the Audible Magic product in use by YouTube, Facebook and others.

This work is intended to be explored in the future as a more efficient method of producing a video or audio content signature in order to add the capability to search for video or audio files that have not been registered (video signature matching, rather than watermark matching).

6 Web-App and Registrar

The Registrar is a simple smart contract which maps content identifiers to blockchain public keys. Implementation details of this are straightforward and are left to the reference implementation.

The Web-App is the front-end module that provides the user interface for both subscribers and nodes to interact with the system. Details of the operation of these components is assumed for the purposes of this whitepaper, but can be found in other Veredictum-based documentation.

7 Indexer and Sorter

The system must crawl the web to understand what files have been added in the previous cycle time period, however, the search space of the entire internet is too large to index within a reasonable cycle time, T_{CYCLE}, e.g. 24 hours. The indexer therefore must take some intelligent measures to reduce this search space. It is intended to use machine learning to derive a probability or relative score for predicting whether a given URL (file) is watermarked.

The profile of users paying for the service is also important to consider so that a subset of users with a high likelihood of being pirated do not dominate the resources of the search network; a poorly designed algorithm may continuously return the same results for a problematic domain that is hosting a large amount of pirated (and watermarked) material, thus providing an 'answer' to the subscriber that is already known, whilst another subscriber misses out on a result that alerts them to a brand new piracy event.

7.1 Indexing of video files on remote servers

Web crawlers are a key component of search engines, and details on their algorithms and architecture are kept as business secrets. When crawler designs are published, there is often an important lack of detail that prevents others from reproducing the work [9].

Additionally, the information revealed by web servers differs from server to server, however some larger social media sites offer APIs so that richer information can be obtained in order to optimise this effort.

Web crawling doesn't require a significant amount of data to be transferred per request, however this grows linearly over the number of unique URLs visited. Tests within our local environment indicate a typical size for an average website to be approximately 500kB for HTML only. Let's assume the metadata we want per URL is around 1/10 of this: 50kB.

If we apply this to the example Facebook search area, we can determine how much data would need to be downloaded, processed and managed in a database. Facebook has around 2 billion active monthly users, so if we assume one URL per user:

$$
\begin{aligned}
&= \textit{(Number of users per month)} \times \textit{(size of URL metadata)} \\
&= 2{,}000{,}000{,}000 \times 50\ \textit{KB} \\
&= 100{,}000{,}000{,}000\ \textit{KB} \\
&= 100\ \textit{TB / month} \\
&= 3.3\ \textit{TB / day}
\end{aligned}
$$

That's quite a lot of metadata in total, however it would not need to be examined every cycle time; many users on Facebook do not upload much by way of video data. The initial parsing of this search space could be fuller, say 1 TB chunks per day for 9 months, before being optimised down. New and high-frequency users would be prioritised and examined more frequently than old and low-frequency users. This would mean that, if resources were constrained for our indexer to between say 0.5TB to 3TB of processing per day, the indexer could improve in efficiency over approximately 12 months.

A combination of a standard web crawler (e.g. `wget`) and browser automation technology (e.g. Selenium) can be used to implement such an indexing module and be extended via platform-specific APIs.

7.2 Indexer management

It is important to ensure that customer resources are fairly distributed with respect to the domains that the Indexer searches (and later the Sorter prioritises). This challenge arises under the assumption that certain users may only be interested in searching those domains that specialise in hosting their type of content. The converse is also true: some domains are only interested in pirating/hosting certain types of content.

This can be managed by growing the search space organically (starting with the major, common sites) and adding additional domains according to user demand; feedback can be gained from the community over time relating to where to look for pirated material (see Manual election below).

7.3 Search space sorting

Machine learning will be used to build a reputation score for URLs within the search space. There are a number of derivable features from the URL itself, metadata, and domain-specific APIs, and these can be used as inputs to train the algorithm. The labelled training data comes from the network itself: if a watermark was found or not, this feeds back to train the neural network. In this way, we can sort the search space before it is allocated to the computation network.

If we take Facebook's URL structure as an example, we can see that it consists of videos having a unique identifier attached to a username: https://www.facebook.com/username/videos/1234567890123456/. Some domains however, such as YouTube, have no user association with the URL for their unique video identifier: https://www.youtube.com/watch?v=t7tA3NNKF0Q, but this is can be derived from the YouTube API. The Sorter will therefore require some domain-specific handling to derive these input values to the algorithm. This will need to be maintained over time as existing APIs evolve and new social platforms emerge. Separate neural networks may be needed for each top-level domain, depending on the data available from each server.

7.4 Manual election

Domains of interest will be able to be submitted by subscribers to be included in the indexer. In order to mitigate the potential attack surface by directing the indexer, the submitted URL can be marked with a high reputation “impulse” that can be tracked over the course of several search cycles in order to assess its accuracy (i.e. whether pirated files were found, and how many, at that URL root). If this provisional impulse was proven to be accurate, then the indexer logic can be permanently updated. A reward for providing accurate information will also be considered in the reference implementation. If it was proven to be false however, this would indicate a malicious actor was attempting to influence the indexer. Manual election of domains therefore constitutes an attack vector and will be explored later in this document.

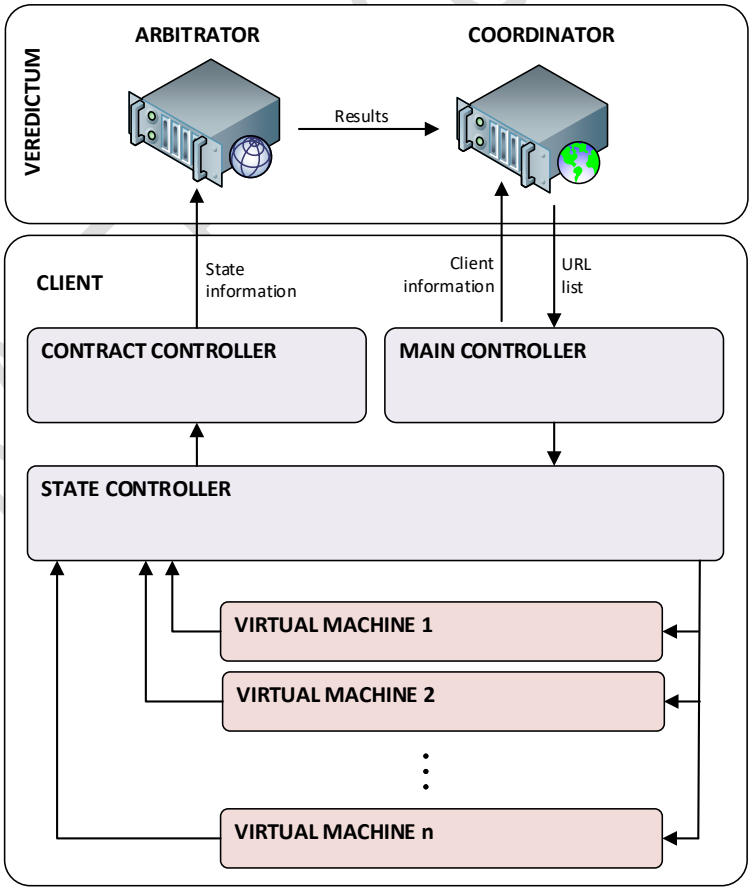
8 Client and Coordinator

8.1 Architecture

The client is composed of the following modules:

- The main controller for client authentication and coordination of work packages.
- The state controller for virtual machine (VM) scaling and delivering files to the spawned nodes for processing, as well as controlling state hashing.
- Smart contract controller for submitting states and final results to the Arbitrator module. This is described as a 'smart contract' however due to the huge amount of data required to be maintained, is initially intended to function on a centrally controlled node. It is also logically separated as it is intended to eventually move this onto a public blockchain.
- A number of virtual machines, based on the capacity of the underlying device and the settings of the user.

A high-level architecture is given below. It outlines the data flow from the Coordinator to a client to the Arbitrator to process, resolve, and verify the content identifiers:



8.2 Main controller

The ‘main controller’ manages the connection to the Coordinator and negotiating participation in the next search cycle, ensuring deposits are held, receiving the URL list, and file downloads.

8.2.1 Client information and connection

8.2.1.1 Client version

All clients participating in the current search cycle must have the same program version. New clients, or clients that need to perform an update, will need to download the latest version of the software and verify its integrity. This can be achieved by comparing the hash of the application with the latest kept with the Arbitrator module.

Ensuring all clients have the same version can be controlled using a pre-defined cycle nonce (i.e. at a particular nonce X, version Y of the client will be used. This is important as all nodes will need to run exactly the same version of the software in order for the nodes within a given subspace to generate the same hash of their internal state.

8.2.1.2 Device capabilities

The main controller will submit device information back to the Coordinator, including details of physical location (country, region, time zone), mainboard, CPU, memory, power supply, network bandwidth and latency.

8.2.1.3 User settings

The main controller will apply some default settings, however the user, on initial setup of the application or at any time after, can modify settings relating to available bandwidth, CPU utilisation (including number of cores, temperature and maximum load), operation times (e.g. operating for 6 hours between midnight and 6am), and custom price of a kWh. A (high) default value can be set based on the location of the client, from approximately $0.10 USD for China to $0.40 USD for Denmark. Note that, just like Bitcoin, regions with cheaper power will be able to generate greater profit.

8.2.2 Deposits

Deposits are required to participate in the network. The main controller will ensure that the user account has sufficient funds before being accepted as a mining node.

The user interface will also enable the purchase of cryptocurrency, or be able to be set so that the initial accumulation of awarded coins goes into the deposit.

8.2.3 Subspace provision

The Coordinator provides a list of URLs that point to a random selection of video files for processing by the miner. Receipt and verification of this list is managed by the main controller. An appropriate source of entropy for this function is left to the reference implementation.

8.2.4 Subspace download

The main controller also manages the download of the video files in the URL list before passing them through to the state controller for injection into the virtual machine(s).

8.2.5 Network size

Nodes will need to be artificially limited by the limited amount of subscription funds available to be disbursed per cycle time. This means that a minimum compensation amount that a node can earn will limit the number of nodes that can join to prevent the awards becoming too diluted. Up until this limit is reached however, the more nodes that are available in the network, the more search space can be covered.

8.3 State controller

8.3.1 Virtual machine scaling

The state controller includes a level-2 hypervisor (also known as a Virtual Machine Monitor – VMM - that instantiates and manages virtual machines) to manage computational scaling. Additional virtual machines (VMs) are instantiated according to both the capacity of the underlying hardware and the user settings. Information about the underlying physical device and user settings (to facilitate the VM controller managing the proper number of instances) are gained from the main controller module.

Part of the role of the hypervisor is to implement standardised virtual hardware upon which multiple virtual machines operate. By ensuring that the underlying hardware presented to the virtual machine is decoupled from the actual hardware, the state controller is able to provide an isolated enclave for the VMs. This reduces the chance that differences in memory state will propagate into Merkle tree mismatches. It also provides a clean interface to serialise and deliver the downloaded files from the provisioned search space into the VM file system.

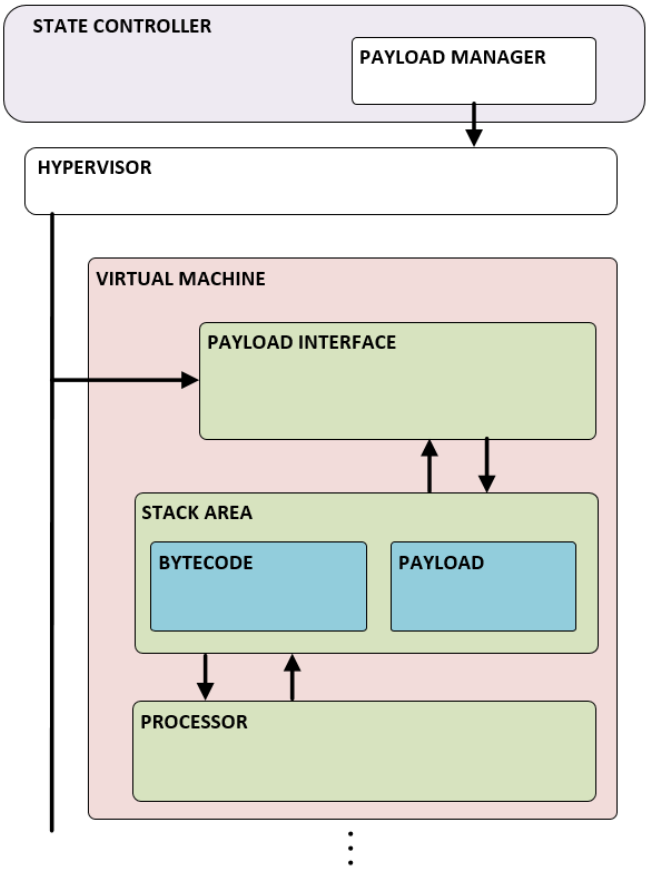
8.3.2 Virtual machine architecture

8.3.2.1 Process vs system virtual machine

As previously mentioned, we propose the virtual machine implementation should be a "process virtual machine" running on a host operating system, similar to the Java Virtual Machine (JVM).

An alternative approach would be to use a minimal Linux operating system running on virtualised hardware, e.g. an x86 architecture running on QEMU. The VM could then run a simple internal stack machine (e.g. Lanai or TinyRAM) and take memory snapshots of the entire VM using the hypervisor. This approach, while able to be achieved with current tooling, is less likely to produce consistent results given the additional complexity of a full operating system interposed between the virtual processor and the client wrapper.

A simple architecture is provided below, showing an expanded view of the virtual machine (running in the user's browser in the case of wasm):



A decoder application that is compiled to bytecode over a native executable will be slower and less power efficient, however research in this area suggests that this will not be significant. Hash-tree-based memory verification in virtual machines experiences some of the same limitations but it is possible to be implemented with an acceptable performance overhead [13].

8.3.2.2 Target

Teutsch and Reitwießner from the TrueBit project have suggested that the LLVM processor architecture would be ideally suited to such an application given the simplicity of the processor

design [10]. However, Google's continued support for the Lanai architecture is not certain, and WebAssembly (wasm) has been suggested as an alternative architecture [11]. wasm is a compiler target that is soon to be compatible with all major browsers. It is being actively contributed to by Mozilla, Microsoft, Google and Apple [12].

Our decoder application source code used in watermark testing is written in C, which is a good candidate for compilation to a WebAssembly bytecode (wasm is designed to run bytecode compiled from C and C++ source files). While no full-featured wasm-based VM's are currently available, several are in active development.

8.3.3 State hashing

For each computational step (or set of computational steps), the client will manage the state hashing of the internal program memory. This will either be a simple hash function or form a Merkle tree, where the Merkle roots will be later submitted through to the contract controller for comparison between clients.

The full state of the machine at the point of forming the Merkle tree or hash must also be saved to disk (including the previous or starting state) in the case of verification reperformance (see sections relating to reperformance design). These internal state snapshots are expected to be between 10 and 100MB each in raw format before compression. The size of these snapshots will influence how many state snapshots are taken in the reference implementation.

8.4 Contract controller

The contract controller receives the state information from the state controller. State hashes are stored within the client logs, however the Merkle roots from the formation of hash trees are further submitted to the Arbitrator. If the Arbitrator is built on a private Ethereum instance in the reference implementation, the controller will also come bundled with an Ethereum API for transaction submission and key (identity) management.

At the conclusion of the computations by the client, a final result is returned to the Arbitrator (the bundled Ethereum client interrogates the Ethereum blockchain state to decide when to submit the final answer). This result may also be encrypted with the public key of the Coordinator to keep this information private.

Note that a peer-to-peer overlay connection is not required between clients; nodes in the same subnet are orchestrated by the central Coordinator and are blinded from each other; they have no way of knowing which other nodes they are being compared against unless they participate in external collusion activities via a private communication channel.

8.5 Results

Once the answer set has been produced by the network, that is, we have resolved the steganographically-embedded content identifiers for the set of files allocated to be searched, it is then compared against the stored content identifiers that are held on the global (blockchain) register. This will then resolve the set of identities that hold the copyrights to those content identifiers.

The Coordinator gathers these final results and reports them to the subscriber base via a web interface. This closes the information loop and effectively forms a telemetry system for their registered content. This process occurs once per cycle time (network search cycle time e.g. once per day).

Users of the service can then take some action on the fact their content has been misappropriated. This could involve issuing an automatic DMCA take-down notice, letter of demand, or general analytics.

9 Arbitrator

9.1 Module vs smart contract

Due to the scale of the coordination exercise proposed by our system, a public blockchain implementation for the Arbitrator functionality is infeasible due to the sheer amount of data needed to be loaded into the smart contract. By the calculation below, assuming that only the 256-bit hashes of the URLs are loaded, the system would require at least 2 GB of storage per day for a search space as big as Facebook. We can calculate this by looking at the number of unique URLs and the size of their tokenised (hashed) form:

$$= (Facebook\ users\ per\ day) \times (size\ of\ URL\ hash)$$

$$= \frac{2 \times 10^9}{30} \times 256\ bits$$

$$= 1.98\ GB$$

Additionally, a public blockchain-based implementation would precipitate a large volume of global transactions. This volume however is not so easily calculated without some implementation assumptions relating to the number of files processed per node. If we assume a worst-case transaction volume where nodes submit a unique transaction containing the Merkle root of their state tree for each URL, the number of transactions per second can be calculated for Ethereum assuming a reperformance factor of 2 and a 15 second block time:

$$= 2 \times \left(\frac{2 \times 10^9}{30 days \times 4 \times 60 mins \times 24 hrs} \right) tx/block\ time$$

$$= 2 \times (11{,}580)\ tx/block\ time$$

$$= 23{,}150\ tx/block\ time$$

$$= 1{,}543\ tx/second$$

This is a very high number of transactions in the current technology environment. Of greater concern is the additional data that would be needed to be loaded into the smart contract: every URL would generate another 2 × 256-*bit* hashes representing the reperformed Merkle roots. This means that at least 3 × 2GB would written to the blockchain per day. Even if this was reduced by a factor of 10 through URL grouping, this is an overwhelming case for a centralised

implementation of the state-matching logic over implementation on the public Ethereum network. The Arbitrator module thus acts similarly to a mining pool, accepting hashes from remote nodes and managing the disbursement of rewards.

While the coordination of state matching is managed centrally, the token value-transfer process can still be maintained on a public blockchain. The mining network would need to trust the central party to make the correct disbursements, however escrow facilities can be used to ensure that sufficient funds are available before mining begins. This reduces the number of transactions needed down to only cover manually-triggered transfers of the tokens into the recipient nodes' accounts, once the states have been resolved.

9.2 Reperformance over cryptography

Distributed computation verification is problematic as there is no cryptographically-efficient way to validate an output O of an arbitrary computation of a known program P with input I [13]. This lack of a cryptographic proof-of-computation is a common problem with 'volunteer' computation found in academia [14]. The verification problem remains an area of active research and in the future, an implementation of an appropriate succinct computational integrity and privacy protocol (SCIPR) may yield more efficient results. As such, computations in our system are **reperformed** at some tuneable verification design factor, $v > 1$. A factor of 1 would indicate no reperformance and a factor of 3 would indicate the computation has been performed 3 times.

9.3 Tuneable reperformance design

The reperformance of computations is done by other nodes within the network. These nodes are matched with one another, at random, to process the same file or set of files. At a reperformance factor of 2, we match, at random, two clients to process the same file set. This reperformance factor is configurable and can also be applied dynamically based on an individual client's trustworthiness (a trust reputation can be applied to all nodes to reduce the total amount of reperformance required in the system as a whole). Indeed, the reputation algorithm can be used to inform the reperformance parameter to less than 2 redundant computations, perhaps down to 1.5, 1.1 or even 1.05 computations (5% reperformance, instead of 200% for two full and independently compared computations).

9.4 Arbitrator module

9.4.1 State ingestion

Verification functionality is ideally implemented in a trusted execution environment due to the sensitivity of incentive handling, however this is infeasible on a public blockchain given the amount of data required to be maintained within the smart contract state. Our system proposes a centralised (trusted) implementation of a smart contract. Given that a blockchain provides an

append-only, time-series state machine, it can be used to reliably record the ordered states of remote computations. Even though the implementation is centralised, the source code can be made freely available and the blockchain database can be freely interrogated via an API. This will enable the program's internal state changes to be verified by external parties and enable a decentralised solution in the future.

The Arbitrator module accepts node state submissions (Merkle tree roots) from multiple clients across the network. Depending on the reperformance parameter for the individual client, these hashes are compared against the hashes of the other matching nodes who submit their results for the same subspace (according to how they were allocated by the Coordinator). Importantly, the Arbitrator also records a hash of the subspace that the nodes are required to process in case a dispute resolution subroutine is invoked; this immutably records the input I into the program P to be recomputed.

If we assume a reperformance factor of 2, we effectively have performed the computation twice and compared the results. In the case that all submitted hashes match, then we assume that the independent nodes have not colluded and no systemic errors have occurred in the client such that a good actor has operated faithfully and the underlying software has not submitted incorrect answers due to a software bug.

9.4.2 State mismatch resolution

If the answers do **not** match, a verification phase is triggered for the nodes involved. The clients are firstly alerted to the fact that one of their virtual machines has been put into a verification mode and is required to reperform a computation under identical conditions (this inhibits for example a client update before this recomputation has resolved). Given that Merkle roots were submitted to the Arbitrator, it can then ask for the submission of all the leaf nodes to resolve where the mismatch has occurred. The Arbitrator can then direct where the virtual machine is required to re-start its computation. This means the hypervisor and virtual machine need to ensure full states have been saved associated with the leaves of the submitted Merkle roots so that it can be reloaded for the verification subroutine.

A binary search is then triggered by the Arbitrator to find the first instance of disagreement between the two nodes. The computation is firstly reperformed between the two states (the first being the state of agreement and the second being the state of disagreement), with N state hashes being submitted periodically in between. The process then repeats until the smallest computation unit is found where the two virtual machines have diverged in their computational results, that is, the two machines agree on the previous state, but not the next state, for a single op-code. This can then either be resolved centrally, or "on-chain" in a trusted execution environment by running the op-code state change via a smart contract, such as an LLVM interpreter implemented in Solidity and running on Ethereum [12]. Full state information could be submitted for every op-code change in a linear fashion, but this would be far more computationally expensive than to

submit $n.log(m)$ transactions, where n is the number of intermediate states, and m is the total number of state changes.

Once the correct client has been found, they are awarded both the original fees, and the deposit of the incorrect client.

The transfer of cryptocurrency is then performed from the deposit made by the Coordinator's account to the solver's, however the withdrawal is delayed by a dispute period. This period is chosen such that there is an acceptably long window of time (e.g. 10 days for a server indexing period of 1 day) to allow a challenge to occur. Because the information that is purported to be correct by the Arbitrator is falsifiable, we can protect good actors from errors in the system as well as prevent bad actors from disputing the process without losing their deposit through the use of a *second* challenge loop. This second round will resolve the dispute (see Dispute resolution).

9.4.3 Dispute resolution

When a challenger triggers a review within the dispute window (this can also occur if the original states were in agreement in the case that the original nodes are suspected of full collusion), the Arbitrator re-runs the resolution routine with the challenger and one of the nodes from the subnet, selected pseudo-randomly.

Like Filecoin's proof-of-retrievability, with our model, we can prove that the nodes have downloaded the correct file(s) as the starting state; all nodes submit their starting state generated from their unique URL list (i.e. submit a hash of the URL list). Even though no computations have occurred at this point, the nodes have cross-validated their starting positions.

The outcome of this secondary routine is expected to be final and authoritative, however we leave any further detail to the reference implementation.

10 Attack Vectors

Careful consideration of the various attack surfaces is crucially important given that the computation work is incentivised; incentivisation opens the system to a range of gamification problems where bad actors seek to receive incentives without having performed the work. This is a key difference to volunteer projects such as SETI@Home [16].

10.1 Sybil attack

Verification of computations is done through reperformance. The reperformance design is based on a set of independent, uncooperating nodes, who are aware that other nodes exist, but have no easy means of communication for the purposes of collusion. A Sybil attack is one in which a single entity spawns an overwhelming number of identities that can cooperate in some way.

This attack is initially mitigated by the random assignment of reperforming nodes by the Coordinator; nodes are not told which other nodes are within its subnet. It is further mitigated through the use of deposits; where the number of collaborating identities overwhelms the network, the dispute subroutine can be instigated with a trusted challenger. If the attack occurred after the system was operational for some period of time, the nodes belonging to the attacking party would be able to be identified due to their low starting reputation and low tenure.

While the deposits of the bad actors would be recoverable, these could be used to pay for immediate centralised processing of their previously allocated subspaces in order to minimise the delay of telemetry data being received by the end customer.

10.2 Metadata tumbling

Web servers hosting pirated material will eventually become aware of our system's efforts to index the material that is uploaded to their site. One way to confuse the system's understanding of what files are hosted on the target web server is to 'tumble' the file metadata to make the files on the web server always appear to be newly uploaded. For example, the file name could be randomised on a daily basis. This could potentially thwart the system as multiple rewards would be paid for the same piece of pirated material (ostensibly a different file, but the same content on the same web server).

One mitigation for this scenario is to tune the reward such that it is more than the cost of compute, but less than the value of time spent to game the system. Thus, the reward per video may need to be tuned such that it is less than the price of the effort to tumble an entire website's metadata, but more than the cost of electricity and compute and bandwidth in order to detect it. Particularly problematic websites may need to be manually removed from the search pool. We leave further mitigations to future work and the reference implementation.

10.3 Encoded stream multiplexing

Where a marked audio stream from one file is merged with a marked video stream from a different file (i.e. different watermarks from different content), this will result in two watermarks being detected. This might be found in the case that a user is appropriating a piece of registered content to produce a fair use ‘mashup’ video.

10.4 Manual election attack

The system accounts for a design which allows users to nominate where to direct the indexer; good actors may become aware of where their content is being illegally hosted and they will have the ability to notify the system to have this included in the indexing algorithm (see section 7.3).

The attack surface here is when a user nominates a location to search but has also uploaded pirated material to that same domain specifically for the purpose of ‘finding’ that material and earning mining rewards.

The design of the incentive scheme is such that rewards are apportioned directly related to computational effort, and not a bounty scheme. In addition, files are allocated at random to nodes on the network so that no one user can directly benefit from this attack vector unless combined with a Sybil attack (see above).

10.5 Cheat default equilibrium

Given that we assume that all actors are rational, we can compute the game-theoretic Nash equilibrium for the case of one file is computed by two nodes. The two strategies are:

(a) Compute the program, spending electricity of 1 unit, and gain an award of 2 units (representing compensation for costs, plus profit); or
(b) Do not compute the program, save electricity cost of 1 unit, earn an award of 2 units.

In the case that the other party cheats, the resolver will determine the correct solver and award the cheater's deposit. Note that the stated deposit amount of "10 units" simply represents a large relative loss. This value will be optimised (up or down) during implementation with further modelling and feedback.

Node A \ Node B	COMPUTE	CHEAT
COMPUTE	1 \ 1	10 \ -10
CHEAT	-10 \ 10	2 \ 2

Thus, we have equilibria at both-compute or both-cheat.

In order to mitigate the both-cheat equilibria, we continue to hold deposits for a 'challenge' period after the end of the cycle time in order to allow third party verifiers to re-verify computations. These verifiers are trusted. They are required to be trusted to ensure that the simple cheat default equilibrium isn't propagated. By ensuring a minimum amount of re-validation (e.g. 20% of all agreeing solutions), over the long term, this effectively ensures that the cheat strategy will result in a loss of deposit for both Nodes A and B:

Node A \ Node B	COMPUTE	CHEAT
COMPUTE	1 \ 1	10 \ -10
CHEAT	-10 \ 10	-10 \ -10

The re-validation and deposit parameters are tuneable per cycle time and will be selected during implementation with further testing.

10.6 General collusion

We can envision a situation where nodes begin their computation with valid state submissions, but then use the submitted state information to find each other on the network, collude via a private channel and finally submit matching bogus hashes for the remainder of the computation, without spending any computational resources.

This may become possible for example if a node accesses the public key (identifier) for their corresponding node, before publicly listing this having a look at the public keys contained in the smart contract, and have an independent web site publish these keys along with their IPs on a separate website. The client can also be examined to extract the URL list which can be used to find the corresponding node on the network.

General collusion can be mitigated in two ways (i) improved blinding (weak), and (ii) second-order verification (strong).

10.6.1 Improved blinding

By using a commit-reveal system for state submissions, an encrypted or randomised version of the state hash can be submitted (e.g. by using a pseudo-random salt, such as the current Ethereum block number). Once both nodes have signalled they have a state hash ready for submission, this can be followed by relatively brief period where the real hash is submitted. Alternatively, nodes can be given a session key-pair, where their answers can be encrypted with the private key, and the public key is kept by the Coordinator, and only revealed after all states have been submitted. Encrypting the URL list however would not be effective as it must be decrypted in order to be used, after which both nodes will be able to find each other via an open matching service.

10.6.2 Second-order verification

By establishing a minimum amount of reperformance over *agreeing* state submissions, we can detect bad actors and claim their deposits in order to pay for the processing of old results. If this factor was x % reperformance over agreeing states, then we would need to hold at least $(100 / x)$ times the cycle reward in a deposit to ensure it is an effective deterrent.

We leave the final details of this to the reference implementation.

11 Conclusion

This paper has outlined a method for implementing a registered content search engine for the purposes of tracking where videos move within a targeted subset of the internet. The method relies on an initial, centralised, file indexing service which examines publicly-facing web servers to determine when new video and audio files have been made available. A distributed network of cooperating computers (nodes) then download and inspect all of these new files to determine which have been watermarked. Results of this process are made available to subscribers of the service who wish to understand where their watermarked material has moved, legally or otherwise.

Given that the distributed computation network is incentivised, we have also presented a mechanism for ensuring the veracity of returned results through the use of deposits and re-validation of computations. Disputes between nodes in the network are managed by a binary-search-inspired [11] subroutine to resolve the divergence of agreement between nodes before computing this diverging step in a centrally trusted execution environment, or via an interpreter on a blockchain.

Appendix B explores the critical economic question of the cost difference between centralised and distributed computation. Detailed business economics of such a system (such as the establishment of the cryptocurrency, service costs, customer pricing, and customer projections) are not considered in this paper, however these are explored in other, related documents.

The high-level design that is presented is able to be implemented in software by anyone, however we will be continuing the development of this system with a ‘reference’ implementation and be adding to this design document over time with a number of technical addendums.

12 About

12.1 About the Author

Samuel Brooks is CTO of Veredictum, leading the design of systems to help combat digital piracy using distributed ledger technology. He is an electrical and software engineer and pursues interests in the theory and application of blockchains and artificial intelligence.

Sam lives in Sydney, Australia.

blog/twitter: samuelbrooks.io
email: sam.brooks@veredictum.io

12.2 About Veredictum

Veredictum is a technology company based in Sydney, Australia. They are developing a digital anti-piracy and distribution platform for video and audio content.

Details can be found on their website: veredictum.io

12.3 Acknowledgements

Thanks to the following reviewers of this whitepaper:

- **Tim Lea** for his endless patience, energy and positivity.
- **Jason Teutsch** from the TrueBit project for his generosity with his time and detailed, critical eye.
- **Age, Luke and Paul** from **Sigma Prime** for their honesty, directness, expertise and friendship, and for finding by far the biggest gaps in the first draft.
- **Robert Allen** for his early and ever-present support and seasoned review.
- **Rhett Sampson** for his enthusiasm, vision, and technical expertise.
- **James Ashburn** for asking every question and his exceptional attention to detail.

13 References

[1] filecoin.io, “Filecoin: A Cryptocurrency Operated File Storage Network,” 2015.

[2] C. Bialik, “Putting a Price Tag on Film Piracy,” 5 Apr 2013. [Online]. Available: https://blogs.wsj.com/numbers/putting-a-price-tag-on-film-piracy-1228/.

[3] Tubular Labs, “The Rise of Multi-Platform Video: Why Brands Need a Multi-Platform Video Strategy,” Social@Ogilvy, 2015.

[4] Kurzsegagt, “How Facebook is Stealing Billions of Views,” 10 November 2015. [Online]. Available: https://www.youtube.com/watch?v=t7tA3NNKF0Q.

[5] L. Overweel, “Stop Freebooting Now,” [Online]. Available: http://stopfreebootingnow.com/#wall.

[6] Facebook, “Facebook’s Top Open Data Problems,” October 2014. [Online]. Available: https://research.fb.com/facebook-s-top-open-data-problems/.

[7] Amazon, “AWS Cost Calculator,” 2017. [Online]. Available: https://calculator.s3.amazonaws.com/index.html.

[8] Wikipedia, “Discrete Cosine Transform,” [Online]. Available: https://en.wikipedia.org/wiki/Discrete_cosine_transform.

[9] Wikipedia, “Web Crawler,” [Online]. Available: https://en.wikipedia.org/wiki/Web_crawler.

[10] G. E. S. D. C. M. v. D. a. S. D. Blaise Gassend, “Caches and Hash Trees for Efficient Memory Integrity Verification,” *Proceedings of the The Ninth International Symposium on High-Performance Computer Architecture,* 2002.

[11] C. R. Jason Teutsch, “TrueBit: A scalable verification solution for blockchains,” March 2017. [Online]. Available: http://truebit.io.

[12] C. Reitwießner, “Reddit - Interactive Verification of C Programs,” 2016. [Online]. Available: https://www.reddit.com/r/ethereum/comments/51qjz6/interactive_verification_of_c_programs/.

[13] ZDNET, “Apple, Google, Microsoft, Mozilla close in on making web run as fast as native apps,” ZDNET, 1 November 2016. [Online]. Available: http://www.zdnet.com/article/apple-google-microsoft-mozilla-close-in-on-making-web-run-as-fast-as-native-apps/.

[14] Ethereum Foundation, “Ethereum Wiki - Problems,” May 2016. [Online]. Available: https://github.com/ethereum/wiki/wiki/Problems.

[15] iEx.ec, “Blueprint For a Blockchain-based Fully Distributed Cloud,” March 2017. [Online]. Available: http://iex.ec/wp-content/uploads/2017/04/iExec-WPv2.0-English.pdf.

[16] Berkeley University, “SETI@Home,” 2017. [Online]. Available: https://setiathome.berkeley.edu/.

14 Appendix A - Background on Verifiability Problem

14.1 Proof of Discovery

Distributed computation verification is problematic as there is no cryptographically-efficient way to validate (cryptographic proof-of-computation), an output O of an arbitrary computation of a known program P with input I [13]. This is known more colloquially as the 'Verifiability Problem'. In essence, how can you trust a remote computer has faithfully run your program and given you back a valid result?

This is a common problem with 'volunteer' computation found in academia [14] and remains an area of active research, known as *succinct computational integrity and privacy protocols* (SCIPR). In order to overcome this, veracity of the results is managed through game theory, where computations in our system are **reperformed** by independent machines and **compared,** with a hefty penalty applied for nodes found cheating.

Hence, our system is a (probabilistic, not a cryptographic) proof of computation, colloquially "Proof of Discovery" as the system is designed to discover registered content.

14.2 Related projects

The following is a list of relevant projects in this area that have been examined in the preparation of this paper:

- SETI@Home
- iEx.ec
- Golem
- Ethereum Computation Market
- TrueBit
- Gridcoin
- LBRY
- LivePeer

15 Appendix B - Electricity Cost Estimate

Assumptions:			
Cycle time	1	days	
Example search space	300	TB	
Assumes all nodes are MacBook Pros			
Total computers sold in the last 6 years	100,000,000	units	
Percentage to assume is in use for this system	0.50%	per cent	
Total computers in use	500,000	units	
total subnets	166,666.67		
Electricity Cost Estimate:			
Estimate a MacBook pro to be around 200W power usage at full utilisation	150	Watts	*sensitive*
Proportion of time used for processing	0.75		
Hours per day running the application	8	hours	
Power use per day	0.9	kWh	
Days per week in use	5	days	
Power consumption per week	4.5	kWh	
Average electricity price in Australia	$0.25	kWh	*sensitive*
Electricity cost per week per node	$1.13		
Number of nodes in subnet	3		*sensitive*
Total electricity cost per week per subnet	$3.38		
Wear Cost Estimate:			
Additional component wear	0.00%		*sensitive*
Average cost of machine per 4 years	$3,000.00		
Additional cost of equipment per week	$0.00		
Total wear cost per week per subnet	$0.00		
Coverage:			
Bulk processing time per day	6	hrs	
Video length multiplier	3	times	
Hours of video processed per week per subnet	90	hrs	

Bandwidth:			
Assumed to be free	0		
Search space (approximate for Facebook):			
Total video uploaded per day	300,000	GB	
File size per minute	0.01	GB	*very sensitive*
Video length per day uploaded	30,000,000	minutes	
Video length per day uploaded	500,000	hrs	
Video length per week uploaded	3,500,000	hrs	
Cost Estimate:			
Subnets needed for the search space	38,889		
Minimum total compensation per day	$18,750		
Minimum total compensation per week	$131,250		
Number of paying customers per cycle time	40,000		
Cost per day	$0.50		
Cost per month	$15.00		
Total value-injection per cycle time	$20,000		
Cost to perform on AWS	$100,000	Approx.	

NOTES:

1. Facebook data estimates found online range from around 300TB to 1,000TB per day for video. The cost for processing more data increases linearly, and we have selected 300TB purely as an estimation point.
2. Total value injection per cycle time has been simply modelled off a fixed cost per customer per day. In the reference implementation, an optimised cost structure would see customers billed per minute of video they have registered, including stratification for corporate customers.
3. Cost of capital expenditure is assumed to be zero (the computer performing the processing is a sunk cost).
4. Cost of managing heat load at the node is also assumed to be zero.