Proof-of-Stream: A Robust Incentivization Protocol for Blockchain-based Hybrid Video on Demand Systems

Yudi Tan
Swanand Kadhe
Kannan Ramchandran

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-42
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-42.html

May 11, 2021

Acknowledgement

I would like to thank Professor Kannan Ramchandran for his guidance over the years and for the opportunity to work on this fascinating project. I would also like to thank Swanand Kadhe for his mentorship and dedication to this project, without whom this project would not have been possible. Finally, I would like to thank Professor Sylvia Ratnasamy for her review and help on the final draft.

Proof of Stream: A Robust Incentivization Protocol for Blockchain-based Hybrid Video on Demand Systems

by Yudi Tan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Kannan Ramchandran
Research Advisor

5/6/2021

(Date)

* * * * * * *

[illegible]

Professor Sylvia Ratnasamy
Second Reader

Proof-of-Stream: A Robust Incentivization Protocol for Blockchain-based Hybrid Video on Demand Systems

Yudi Tan
University of California, Berkeley
yudi98@berkeley.edu

Swanand Kadhe
University of California, Berkeley
swanand.kadhe@berkeley.edu

Kannan Ramchandran
University of California, Berkeley
kannanr@eecs.berkeley.edu

***Abstract*—**
The rise of the Video on Demand (VoD) industry led to unforeseen technical challenges, ranging from scalability issues to network bottlenecks. As VoD platforms increasingly rely on large content delivery networks (CDN) to lower the bandwidth requirements on the origin servers and to improve latencies, we observed that the streaming costs are still unbearably high. In response to these challenges, in recent years, hybrid VoD architectures started gaining traction both in academia and in industry as an alternative to traditional CDNs. However, amongst all of the systems which we've analyzed, to the best of our knowledge, none of them are robust enough to prevent collusion-based attacks against stakeholders, and this limits the viability of these solutions. In this paper, we present the design and implementation of a robust incentivization protocol for blockchain-based hybrid VoD systems, Proof-of-Stream, which we argue is the critical missing piece of the many proposed hybrid VoD systems available today.

I. Introduction

Not too long ago, linear broadcasting schedules and theatrical releases were in control over the content that was available to viewers. However, over the last few decades, there has been a paradigm shift where an increasing amount of entertainment content is now made readily available for consumption in an on-demand fashion. As an increasing amount of digital media content is now offered through platforms such as Netflix, HBO, Amazon Prime and Google Play, consumers have more control over what and when to consume. Nevertheless, the rise of the new video-on-demand (VoD) industry is accompanied with unforeseen technical challenges. As the quality of video content improves, more storage and bandwidth are required to serve such streaming sessions. In a recent technical report produced by Cisco [1], it is stated that video accounts for over 82% of internet traffic in 2020. In order to meet the ever growing demands of VoD content, companies typically have complex distributed video ingestion and serving systems to meet these new technical requirements. Moreover, to ensure low streaming latencies for users located all over the world, companies often employ content distribution networks (CDN) such as Akamai to cache and serve content assets through the edge. Despite so, bandwidth is often still the bottleneck of such systems and can result in high costs for VoD companies due to the sheer amount of data each origin server has to stream to clients. Moreover, as bandwidth becomes the bottleneck, it becomes increasingly harder to serve content to viewers in developing regions who do not have access to expensive public infrastructure and CDN support.

In response to these challenges, in recent years, hybrid VoD architectures have started gaining traction both in academia and in industry as an alternative to traditional CDNs [16] [17] [18]. Such hybrid systems, which will be described in detail in Section II-A, typically employ edge devices as cache nodes (often co-located on user devices/nodes) to significantly reduces the load on the origin server while also enabling low-latency and cost-efficient streaming in rural areas. These systems are also typically integrated with public blockchain infrastructures like Ethereum to enable more trust between stakeholders and to automate payment functionalities such as royalty management in a decentralized way [16][17]. However, there are two main issues with existing systems. Firstly, none of these systems, to the best of our knowledge, are robust enough to prevent collusion-based attacks against stakeholders. Secondly, in order for such systems to be viable at all, a robust incentivization protocol built to protect the interests of stakeholders such as artists, advertisers and cache nodes is critical for adoption.

In this paper, we present the design and implementation of a robust incentivization protocol for blockchain-based hybrid VoD systems, Proof-of-Stream, which we argue is the critical missing piece of the many proposed hybrid VoD systems available today. Our implementation of the protocol in the paper is built on top of an existing hybrid VoD system, CalVoD [2][3][4][5][6], developed in the University of California, Berkeley. The design, however, is platform-agnostic and can be easily implemented and integrated with any hybrid or decentralized VoD platforms.

In the following sections, we aim to present several core building blocks and primitives which are helpful to implement a robust incentivization layer for blockchain-based hybrid VoD systems. Specifically, we propose a robust incentivization and view count guarantee scheme for such systems by leveraging smart contracts to keep track of view counts and user purchases of VoDs.

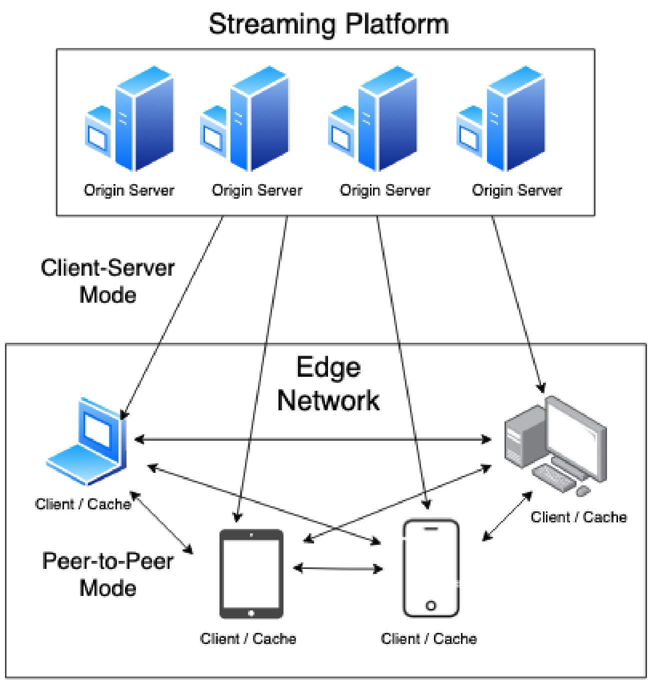


Fig. 1: A typical hybrid VoD architecture consists of a set of origin servers owned by the streaming platform as well as a set of edge devices consisting of users and cache nodes. User clients stream from origin servers via client-server mode whereas they stream from neighboring cache nodes in a peer-to-peer fashion.

The rest of the paper is organized as follows. Section II describes the Hybrid VoD Architecture as well as Blockchains and Smart Contracts, providing readers with necessary background for understanding the remainder of the paper. Section III introduces and analyzes existing works in the space, as well as how they fail to provide an incentivization protocol robust against many common collusion-based attacks. Section IV describes the system architecture. Section V introduces the threat models we've considered in our design as well as our overall design goals. We then introduce and describe in-depth our proposed protocol in Section VI. In Section VII we evaluate our protocol's robustness and deployment costs. Finally, we conclude in Section VIII.

II. Preliminaries

A. Hybrid Video on Demand Architecture

In recent years, there have been several attempts to reduce the streaming costs of VoD systems by employing a hybrid architecture[16] [17] [18]. Whereas in a traditional VoD system the origin servers handle and serve all user requests, a hybrid VoD system (as illustrated in Fig. 1) has additional cache nodes on the edge (often co-locating with user nodes), which can serve other user requests in a peer-to-peer (P2P) fashion, therefore reducing the overall load on the origin servers.

In a hybrid architecture, there are typically four groups of stakeholders.

- *Origin servers:* these are regular content-streaming servers owned by the streaming platform.
- *Users:* they connect to the network and pay for content
- *Cache nodes:* they are located on edge devices whom are often also users who serve neighboring users' stream sessions by streaming content packets in a peer-to-peer manner, reducing the overall load on the origin servers
- *Content creators:* these are artists who provide content to the streaming platform and receive royalty in return
- *Advertisers:* they pay the streaming platform to broadcast their advertisements.

In this hybrid setting, the bandwidth requirements on the origin servers can be drastically reduced since content packets of a particular stream session can be "reused" and forwarded by a cache node (which is typically just another user who has that content packet already cached) to a nearby user node which is requesting for the same content. The origin server only streams directly to clients when cache nodes are not available to fully fulfill a user request. As the user base increases, a hybrid VoD system will drastically decrease the load and costs on the origin servers since more content packets will be available in the cache nodes, which can then be used to serve new requests in a P2P manner.

Traditionally, advertisers and content creators are regarded as separate stakeholders. However, we realize that a growing trend in recent years in the Asian VoD market is that advertisements are increasingly being directly embedded into content. Specifically, there is a trend for "creative embedded advertisements" [1] in VoDs where these advertisements share the same casts and story lines as the actual content and are directly embedded, rather than pre-rolled, into the VoDs. For the purposes of this paper, we will mainly consider this group of advertisers and show that our protocol will help simultaneously protect the interests of both the artists and these advertisers. In future work, we hope to explore techniques which are more general, to protect the interests of different types of advertisers.

From our experiences with deploying and running CalVoD, we've identified the key building blocks common in such hybrid systems:

- Incentivization mechanisms to encourage users to simultaneously act as cache nodes
- Artist royalty-management module
- Advertisement fee-management module

As demonstrated by various Blockchain ecosystems such as Bitcoin [7] and Ethereum [8], an incentivization mechanism is crucial for adoption and scalability. Since CalVoD also requires a huge number of independent (cache) nodes to be practical, an incentivization mechanism is crucial. Moreover, as a VoD system which depends heavily on independent partnerships with artists and advertisers for revenue, CalVoD requires several modules to pay artists and charge advertisers fairly. In this paper, we define a fair payment as being proportional to the true view count of the underlying content.

While implementations of royalty and advertisement fee modules can vary across systems, in this paper we aim to introduce several core primitives which can aid practitioners in implementing these modules in practice.

B. Blockchain

Blockchain is the technology underlying many emerging cryptocurrencies like Bitcoin [7] and Ethereum [8]. It allows users to exchange value in a truly decentralized and peer-to-peer manner, without depending on a central organization or authority to oversee and process the transactions. From a datastructures point of view, the blockchain is an immutable, distributed and append-only ledger of linked blocks, where each block records a set of transactions. These blocks are cryptographically and chronologically linked, hence the name "blockchain" [9]. Within a blockchain ecosystem, there is typically a set of nodes called miners, which run a consensus algorithm and are responsible for validating and recording transactions to the distributed ledger.

Besides being decentralized, blockchains also provide several interesting features that our system depends on in order to be functional:

- *Transparency* since all transactions which are recorded on the blockchain are visible to all nodes in the network.
- *Liveness* since participants can always reach the blockchain while more transactions are continually being processed and new blocks being added.
- *Blockchain address* which each participant in the network has. This address is bound to the hash of each user's public key and cryptography guarantees that only the holder of the secret key can send or sign messages corresponding to this public key [10] [11].

C. Smart Contracts

Smart contracts are user-defined programs that can be deployed and ran on a blockchain such as Ethereum[8]. Once deployed, users are able to interact with these contracts, which will cause nodes currently running these contracts to update their local replicas according to the execution results. In order to interact with the smart contract, such as to invoke a method, the caller has to pay some amount of fees also known as gas. The amount of gas required for each type of execution is listed in Appendix G of the Ethereum Yellow Paper [9] and is reproduced in Appendix A.

In our system, we use smart contracts to store persistent and dynamic state, as well as to facilitate the automatic payment-splitting between various stakeholders. Just as any other transactions on a blockchain, the states in smart contracts are visible to all participants, allowing for easy audits. Moreover, smart contract source code will also be open-sourced, hence any participants can verify the logic of the smart contract at any point in time.

III. Related Works

Here we analyze several existing hybrid VoD solutions and discuss how a scheme like PoS can help complement them.

A. CalVoD

CalVoD [2] is the hybrid video-on-demand system which we began our research on. It utilizes edge devices as cache nodes to serve user requests for video-streams in a P2P fashion, therefore achieving a higher quality-of-service for the user while also reducing costs on the origin servers. However, CalVoD itself is primarily focused on the streaming layer, and from our experiences with deploying and testing it, lacks an incentivization scheme to make it practical for production. As such, this paper is the result of a year of designing and implementing a robust incentivization protocol to enable CalVoD to be a practical hybrid VoD system. Our protocol has since outgrown CalVoD and is platform-agnostic.

B. MyTVChain

MyTvChain [17] is a blockchain-based hybrid VoD system targeting the sports media industry. Its system architecture mirrors that of the original CalVoD design which consists of an origin server as well as a set of edge devices acting as cache nodes (with co-located users). A typical stream request will be fulfilled and served entirely by neighboring cache nodes if possible, or else, by the origin server. A blockchain module then handles royalty management for the content

[1] The exact term for this is described in a Chinese wiki: "https://baike.baidu.com/item/创意中插" We can't seem to find English literature on this perhaps due to how new this trend is; however this seems to be an extreme form on the "product placement" marketing / advertising strategy typically employed.

creators (sports clubs in this case) as well as cache payments. However, despite having this incentivization layer, the authors do not describe any threat models their system guards against and therefore is unclear whether they are resilient and robust against the many collusion-based attacks common in such systems, as further described in section V.

C. Aurum

Aurum [16] is another blockchain-based hybrid VoD system which builds on top of CalVoD. The paper covers a broad range of topics ranging from techniques for content-correlation to providing quality of service (QoS) through the use of hybrid architectures. However, the paper lacks discussion of the incentivization mechanism, which from our experience, is a core functionality necessary for the platform to be practical and sustainable in the long run. As such, we think that a robust incentivization protocol like Proof-of-Stream is the missing piece that will help complete Aurum.

D. Livepeer

Livepeer [18] is a decentralized blockchain-based media server implementation for livestreams. Its goal is to provide a decentralized platform for common media operations such as transcoding and transmuxing. At a high level, broadcaster connects to the blockchain network and requests for a livestream service (i.e. to transcode its live stream packets and stream to requesting users). Then a sophisticated and secure process selects a worker node to transcode the packets and then finally records the proof of transcoding onto the blockchain. At the end of the process, various stakeholders receive their fair share of payments. Though the whitepaper presents quite a robust solution[2] to the video-processing process, the authors noted that the design and implementation of a robust streaming phase is left for future work. As such, again, we think that our proposed scheme can complement their solution really well, resulting in a robust end-to-end hybrid media and streaming server implementation.

IV. System

In this section, we describe the overall architecture of our system. We first introduce the network and market models under consideration. Then, we describe the overall architecture of our system, and finally provide a detailed explanation of each component.

A. Network Model

Our system's network model (Fig. 2) consists of the following entities:

- Hybrid VoD Network. This network is where users, cache nodes and the origin servers communicate. Fig. 1 illustrates the typical communication patterns between the different entities. Users request and pay for content by communicating with the origin servers in a client-server mode. Users also communicate amongst themselves in a P2P mode when acting as / being served by caches co-located on user nodes. In this paper, we will treat CalVoD's network model as a black-box and avoid diving too deep into its actual implementations since real-world VoD systems are each implemented very differently.
- Blockchain. At the core of this paper is the integration of a blockchain network to the CalVoD network. The blockchain network will handle the user-payment, royalty-management, view count-tracking as well as cache-incentivization through the use of smart contracts.

[2]Here we refer to robustness in the game theoretical sense

To sum up, user sessions and content packets are streamed between the origin servers, cache and user nodes over the hybrid VoD network, while payments and custom metadata specific to our proposed protocol (Section VI) are communicated and recorded over the blockchain network (illustrated in Fig 2).

B. Market Model

The system described in our paper targets a niche business model. While there are many forms of VoD services out there in the market, including subscription-based VoD (SVoD), transactional VoD (TVoD), advertisement-based VoD (AVoD) and other hybrids, this paper, and CalVoD, primarily focuses on the market space of a hybrid of TVoD and AVoD where advertisement fees as well as a pay-per-view model dominates the income for the VoD service. Specifically, our proposed scheme is most appropriate for VoD systems which employ a pay-per-view model where a user's payment to rent or purchase a content is then further divided between various stakeholders such as the streaming platform and the artist. In addition, we also learned that in many VoD platforms in Asia, pay-per-view content often also contains advertisement fragments embedded within the content itself. As such, the view-count generated by each of these content is also directly proportional to the advertising fees. Hence, our system assumes that such advertisers will also contribute to the overall revenue of the platform and therefore it is important for our scheme to protect their interests.

C. System Architecture

The system (Fig. 3) includes five components: Origin Servers, Caches, Users, Content Creators and the Blockchain. Advertisers are also part of the system; however, they are not included in the diagram because interactions between the platform and the advertisers typically happen off-chain via private payment agreements. In the following section, we briefly explain the dynamics of the system, and defer the protocol description and implementation details to Section VI.

When a user joins the system, it connects to the hybrid VoD network, which we treat as a black-box model. In Section VI, we will introduce a complete end-to-end system by building on top of CalVoD, though it is important to note again that the scheme and interfaces introduced later are generic enough to be implemented on any existing hybrid VoD system with minimal changes needed to the underlying VoD system. Once the user is part of the VoD network and decides on a content to purchase, they will pay for the content by sending a transaction to a smart contract corresponding to the content they have

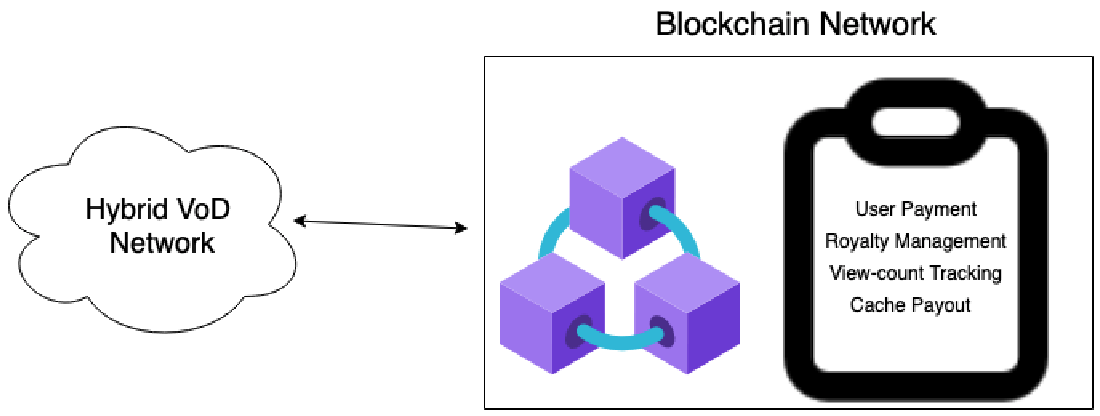


Fig. 2: Network Model

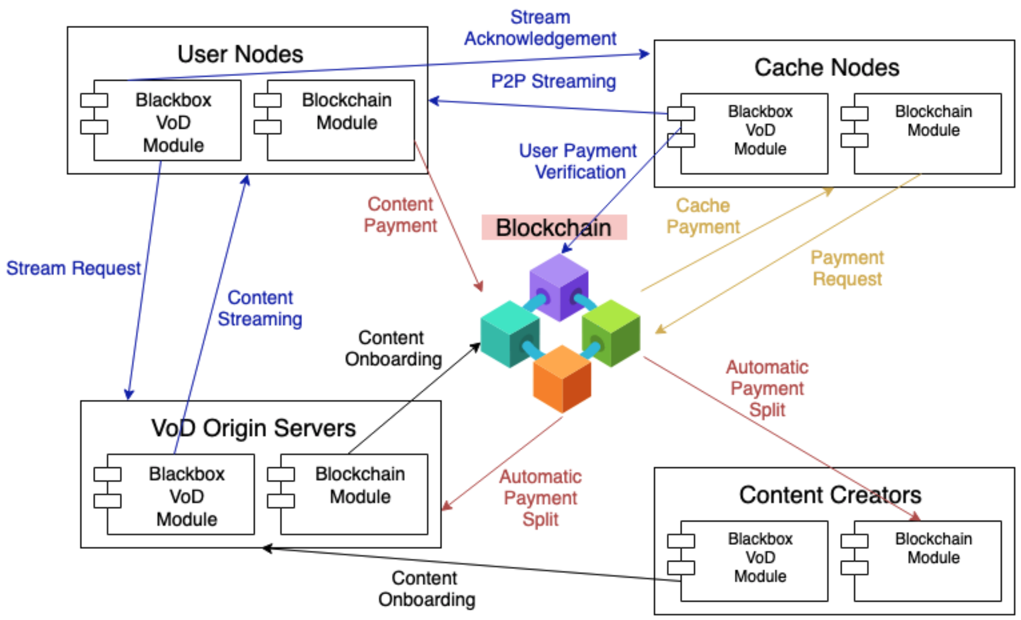


Fig. 3: System Architecture Diagram

chosen (i.e. each content on the VoD platform has its own smart contract counterpart which handles payments and other features). The user then provides the transaction hash to the VoD network, which will validate the user's payment. Once validated, streaming begins on the VoD network. In other words, our system works by extending hybrid VoD systems with a blockchain interface which handles payments as well as other features such as view-count tracking.

V. Threat Model and Design Goals

In this section, we introduce the adversarial assumptions and design goals of our system.

A. *Adversarial and Threat Model*

In our design, we are mainly concerned with collusion-based adversaries which try to compromise the stability of the system. In our set-up, the various parties involved include the server, users, cache-nodes as well as artists. As such, there are various possible collusion scenarios, which form the basis of our threat model. In this section, we will introduce each of these potential threats as well as discuss ways to prevent and mitigate them.

1) Server-User Collusion Against Artists: The first potential threat this system faces is a collusion between the origin servers and the user. One main motivation underlying such a collusion is to reduce costs, both for the server and for the user. The colluding parties in this case are incentivized to stream and pay outside of the proposed scheme since royalty fees can be avoided and thus lowering costs for both the server and the user. Specifically, such a collusion happens whenever a user colludes with the origin server to begin a stream-session without paying through the smart contract. Instead, the user pays the server directly off-chain to initiate a streaming session, instead of following the protocol and initializing a streaming session by first paying the smart contract.

Such a collusion will result in the content creator receiving a less-than-fair share of payment since the user's payment to the server bypasses the smart-contract which handles royalty payment to the content creator. In addition, since the smart-contract is not invoked for this stream session, the view count of this stream session is also not recorded. Referring to Fig. 3, this can be visualized by removing all the red arrows.

We now show that such a collusion is unlikely and irrational. The goal of distributing video chunks onto independent cache-nodes is to reduce the load and streaming costs on the origin server. Assuming that cache nodes are honest, any user involved in the collusion will only be served by the origin server since honest cache nodes following the protocol will check for a valid user payment first (illustrated as the "User Payment Verification" arrow in Fig. 3) before streaming. Therefore, such a collusion reduces the system to a centralized VoD system where the origin server is the only node servicing the user's stream request. This results in an increased streaming cost and burden on the origin server, which undermines the benefits of the hybrid VoD architecture; therefore, such a collusion is irrational from the origin server's point of view.

2) "Streamer"-User Collusion Against Artists: The above collusion can be extended to include colluding cache nodes. "Streamers" in this case refer to nodes which can serve user requests, and in our set-up, they refer to a set consisting of the origin servers as well as several colluding cache nodes. Such a collusion happens when the origin servers, together with several colluding cache nodes, decide to stream to a user who has paid them directly off-chain, bypassing the smart contract. In this case, the artist is again the victim, as they will not receive their fair share of payment for this particular stream session. Again, we can show that this collusion will not work well in practice. CalVoD cache nodes each has very little video chunks available to stream to users, and therefore a large number of independent cache nodes are often required to fully serve a user's stream request.

Assuming that most cache nodes are honest (otherwise, this system reduces to a piracy network), such a collusion will necessarily result in an incomplete and/or low-quality stream from the user's point of view. Specifically, a colluding user's stream sessions will only be served by the origin servers as well as the set of colluding nodes since honest nodes will verify the payment to the smart contract before initializing the stream. As such, the quality of stream received by the malicious user will be proportional to the number of nodes participating in the collusion, which can be assumed to be very low.

3) Dishonest Origin Server Against Advertisers: Since advertisement fees, on top of user payments, make up a large portion of the server's revenue, a natural threat in our system is a dishonest server acting against the advertisers. As advertisement fees are often proportional to the view count of the content (which has the advertisement embedded), the server is incentivized to inflate this view count to overcharge advertisers. In our design, the view count is a state tracked within the smart contract corresponding to the underlying content. As we will describe fully in Section VI, the only way to increment the view count in the smart contract is to send a payment to the contract. Hence, in order for the server to inflate the view count, the server has to pay the smart contract, just like any other user. From this point of view, this scenario is no longer considered a threat since the server is just acting as a regular user paying for the content and incrementing the view count. Therefore, a decision to inflate the view count of a particular content in order to increase advertisement revenue boils down to the net margin of such an act. In other words, such an "attack" is only rational if and only if the advertisement fee per view received by the server is greater (highly unlikely) than the royalty cost per view paid to the content creator.

B. Design and Functionality Requirements

Under the aforementioned system model and adversarial assumptions, the goal of this paper is to develop a robust incentivization mechanism on top of CalVoD with the following design goals:

1) Ensure an accurate view-count for each VoD: This is the most fundamental requirement for our system. At its core, the mechanism should be able to faithfully track the true view-counts for each VoD available in the catalog, as the view-count will be directly proportional to the income received from the advertisers as well as to the royalty fees paid to artists.

2) A modular design which can be easily integrated into existing hybrid VoD systems: Though we present our design and implementation on top of CalVoD, our goal is to have our proposed scheme be platform-agnostic and usable in any hybrid VoD setting which shares similar attributes to CalVoD.

VI. Our Proposed Scheme: Proof-of-Stream

In this section, we will present our design and implementation of the Proof-of-Stream incentivization layer on top of CalVoD, while also describing how our implementation is platform-agnostic. Proof-of-Stream (PoS) consists of the following distinct phases: content on-boarding, content stream request, payment splitting, streaming and cache acknowledgements, and cache payments. These stages are visualized in Fig. 3. In each of the following sub-sections, we will detail each phase in detail as well as provide pseudo-code wherever necessary to demonstrate the ease of integration of Proof-of-Stream into any existing VoD systems. We will also attempt to make it explicit that the following scheme is platform-agnostic.

A. Content On-boarding

In this section, we discuss how CalVoD on-boards a new content. For the purposes of this paper, "on-boarding" a content refers to the process of getting a content ready for the Proof-of-Stream layer, rather than the process of ingesting a content from an external source into CalVoD's internal storage layer (i.e. not referring to the encoding, transcoding, decoding and storage pipeline).

```
1  contract ContentSmartContract {
2      string public title;
3      address payable[] public artists;
4      uint256[] public artists_percentages; // NOTE: the index has to match those in the artists field
5      address payable public origin_server;
6      uint256 public content_price; // NOTE: this is in Wei, not ether, since msg.value in PayForStream is in Weis.
7      uint256 public views;
8
9      constructor(string memory Title, address payable[] memory Artists, uint256[] memory Artists_percentages, uint256 Content_price) public {
10         title = Title;
11         artists = Artists;
12         artists_percentages = Artists_percentages;
13         content_price = Content_price;
14         origin_server = payable(msg.sender);
15         views = 0;
16     }
17
18     function PayForStream() public payable {
19         require (msg.value >= content_price);
20         uint256 moneyToReturn = msg.value - content_price;
21         payable(msg.sender).transfer(moneyToReturn);
22         uint256 total = 0;
23         for (uint256 i = 0; i < artists.length; i++) {
24             uint256 current_payable = (content_price * artists_percentages[i]) / 100;
25             total += current_payable;
26             artists[i].transfer(current_payable);
27         }
28         origin_server.transfer(content_price - total);
29         views += 1;
30     }
31 }
```

Listing 1: Proof-of-Stream Smart Contract Code

Whenever a new content, con_1, has been ingested into CalVoD's system, we need to on-board the content onto the Proof-of-Stream layer as well. To do so, CalvVoD will first need to deploy a content smart-contract, csc_{con_1}, onto Ethereum. csc_{con_1} will be specific to this particular content, and will be responsible for keeping track of the views and purchases of the content as well as be used for splitting the user payments to the various stakeholders (e.g. royalty management). As shown in the smart contract code above (Listing. 1), each of these content smart-contracts' initialization will require certain parameters such as the Ethereum Account Address of the various stakeholders of the content as well as the percentage of each payment they should receive (i.e. royalty percentages etc). Once created, csc_{con_1} will have an initial view count set to 0, its origin server contract address pointed to CalVoD's Ethereum Account Address and its stakeholder addresses and payment percentages correctly initialized as well. Upon deployment of csc_{con_1} onto the main net, we will receive its deployed contract address, which will be given to users later on for them to pay and initialize stream sessions. From this point onward, csc_{con_1} will be the smart-contract representation of the content, where view counts and stream payments will be processed, and this concludes the on-boarding process for this content. For any existing VoD platforms wanting to integrate Proof-of-Stream into their existing systems, they will need to perform a backfill procedure by creating and deploying a content smart contract csc_{con_i} for each and every existing content, con_i, in the VoD platform's catalog and storing each of these deployed contracts' addresses in a database. In other words, there needs to be a mapping for every content available in the platform's catalog from actual content to their smart-contract representations on Ethereum (Fig. 4).

B. Content Stream Request and Payment Splitting

In this section, we introduce the process of requesting and paying for a stream session of a particular content, con_1. In a pay-per-view model, when a user is connected to the CalVoD network and decides to stream con_1, they will request for the smart contract which is responsible for that particular content, csc_{con_1}. CalVoD fetches the contract address of csc_{con_1} and forwards this contract address to the user, where the user can send a payment to by using the contract's PayForStream method. As Ethereum provides an event API which allows events to be sent to listeners whenever transactions to a smart contract are mined, CalVoD can set up such a listener such that when the user payment transaction has successfully been mined, CalVoD can begin its streaming session to the user. As shown in Listing. 1, PayForStream will increment the view count state of the contract (line 29) while also splitting the user's payment to send to the various stakeholders (lines 23 to 28). For instance, for a contract which is initialized to have the content creator A receive 30%, content creator B receive 40% and server receive 30% of the payments, a user's payment for a 10 Ether content to the PayForStream method will result in sending content creator A 3 Ether, content creator B 4 Ether and the origin server 3 Ether. Once the payment transaction has completed, the transaction's address is made available to the user both as a form of "receipt" (proof of payment) as well as to be used by CalVoD nodes to verify payment.

Since Ethereum contracts' public state variables do not expose setters, and that our smart contract does not expose such setters, there is no way to update the view count state in the variable through external means. The only way we can increment the view count state is through our exposed PayForStream API, and therefore we can guarantee that no malicious users can alter the view count without first paying for the full price of the content (and even by paying the full-price, the user is only able to increment the view count by 1, which is accurate). This aligns with our design goal of

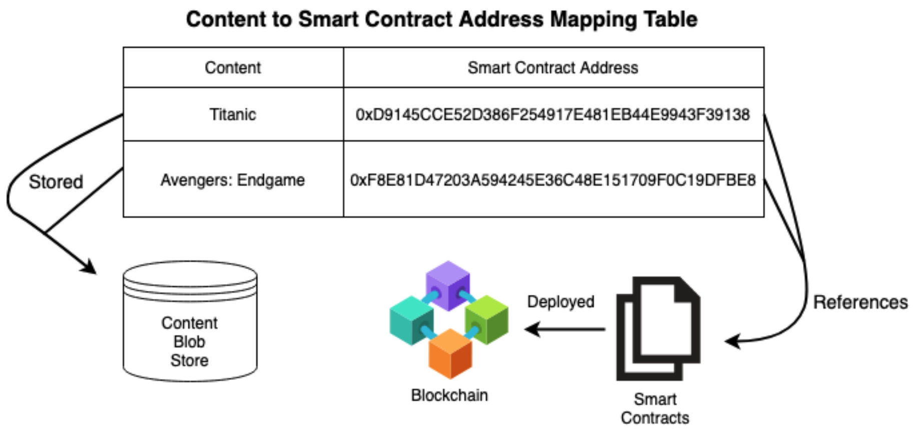

Content to Smart Contract Address Mapping Table
Content
Smart Contract Address
Titanic
0xD9145CCE52D386F254917E481EB44E9943F39138
Avengers: Endgame
0xF8E81D47203A594245E36C48E151709F0C19DFBE8
Stored
References
Content Blob Store
Blockchain
Deployed
Smart Contracts

Fig. 4: Each VoD system using Proof-of-Stream will need to store a mapping between the contents available for stream and their smart contract counterparts.

ensuring that view counts are accurate, since each payment is linked to exactly 1 view count, as desired. Furthermore, since the payment is handled entirely by the smart contract rather than some black-box mechanism, artists and advertisers have full transparency over the payment mechanism and true view-count, and therefore can be guaranteed to receive their fair share of payments. The smart contract's source code is also openly available for audit.

C. Content Streaming and Cache Acknowledgements

Once a user has paid for a content via the PayForStream function on the content's corresponding smart contract, they can begin their stream session. In the CalVoD ecosystem, this means that the user can now connect to the CalVoD network and request a stream session by providing their payment's transaction address from the previous section, as a proof of payment. Upon validating the payment, CalVoD will begin notifying the network of available cache nodes to service this user's stream request. For the purposes of this paper we will not attempt to describe the in-depth process of the stream session since each VoD platform has its own streaming layer with different implementations and the Proof-of-Stream layer described in this paper works with any underlying streaming implementations. The only relevant concept here is that the VoD system will need to verify the user's payment, using the transaction hash returned in the previous step, before streaming to the user. During the streaming session in CalVoD, we also require the user to periodically send acknowledgement packets back to the caches which served their stream session. Cache acknowledgement here is mainly used by CalVoD to pay its cache nodes fairly, though this is optional in other hybrid VoD systems which employ different black-box payment models to its cache nodes.

D. Cache Payments

This stage is optional, depending on the type of payment model employed by the underlying VoD system. In our implementation, CalVoD pays its caches an amount proportional to the amount of bandwidth and packets it provides to other users. In exchange for streaming content to users, cache nodes will receive stream acknowledgement packets signed by users. Cache nodes can then exchange these acknowledgement packets with the CalVoD origin server for payments. In this paper, we treat and delegate the cache-payment process to an off-chain service, since we acknowledge that there can be a variety of business models available, each having their distinct way of paying cache nodes. However, a realistic and simple approach for on-chain payments would be through custom ERC20 tokens. In such a set-up, we envision that users will receive some amount of these tokens from the CalVoD server upon purchasing a content. Then, instead of acknowledgement packets, the user will transfer some of these tokens over to the cache nodes which served their streaming session. Finally, cache nodes can exchange these tokens for monetary payments from the CalVoD servers. We hope to fully iron out the details of this fully decentralized setup in future work.

VII. Evaluation

In this section, we evaluate our system through several angles. First, we revisit the threat models and design goals introduced in Section V and evaluate our design based on those goals. Then we examine the modularity and costs of integrating Proof-of-Stream with an underlying hybrid VoD system. Finally, we evaluate the execution costs, mainly focusing on the costs associated with using Ethereum as the blockchain layer.

A. Threat Evaluation

As introduced in earlier sections, the goal of the PoS layer is to provide a robust incentivization scheme for hybrid VoD systems, focusing on stakeholders protection. In Section V, we introduced the various collusion based attacks which may possibly undermine the viability of such hybrid VoD systems, as well as arguments for why those attacks are irrational and unlikely under PoS. Here, we reiterate some of those key ideas.

1) Collusion-based Attacks: Such attacks typically involve a colluding party of origin servers, cache nodes and users against the content creators. These colluding parties are incentivized to transact and pay off-chain, bypassing the content smart contract, so that they can lower their costs by not paying the associated royalty fees. Assuming that a majority of the cache nodes are honest, we argue that such attacks are irrational and unfeasible in practice under PoS since a hybrid VoD system served by a colluding server and a small set of colluding caches will be unable to produce high-quality streams with low streaming costs. In fact, if the user ends up receiving a high-quality stream, this necessarily means that the majority of the stream burden must have been on the origin servers, therefore reducing this setup to that of a non-hybrid centralized VoD system. Hence, for the VoD server to reap the benefits of a hybrid architecture (e.g. lower streaming costs and greater quality of service), it must rely on a huge number of independent cache nodes, which we can assume to be honest.

2) View Inflation Attack: The VoD platform is also incentivized to increase profit margins by overcharging advertisers. As advertisers pay the VoD platform an amount proportional to the view count of the advertisement (which in our set-up is equivalent to the view count of the content in which the advertisement is embedded), the server is incentivized to inflate this view count. In our scheme, the only way to increment the view count of any content is by invoking the PayForStream method on the content's corresponding smart contract. Since this method requires the caller to pay the full amount of the price of the content, the VoD server has to pay for the content for each view count it tries to inflate. Doing a simple analysis yields that as long as the advertisement fee for a single view is less than the per-view royalty fee paid to the content creators, which is the typical case, a server's decision to inflate the view count is irrational and hence unlikely in practice.

B. Integration Costs

Beyond preventing the adversarial attacks described in our threat models, another goal of our system is to be as modular as possible so that it can be integrated into any existing hybrid VoD systems. Our experience with implementing PoS and integrating it with the CalVoD streaming system shows that the integration effort required is minimal. In order to integrate an existing hybrid VoD system with Proof-of-Stream, all that is needed is for the hybrid VoD system to implement the following modules (Fig. 3)

- *Blockchain Module within Origin Servers*- This module handles deploying new content smart contracts for each corresponding content in the VoD system's catalog.
- *Blockchain Module within User Clients*- This module handles user payments by encapsulating users' calls to the PayForStream API in content smart contracts.
- *Blockchain Module within Partner Clients*- This module handles receiving royalty-payouts for partners including content creators.

Specifically, whenever a new content is ingested into the underlying VoD system, the blockchain module within origin servers will need to onboard the content onto the blockchain. Then, whenever a user decides to purchase a content, the blockchain module within the user's client can handle the payment.

C. Execution Costs

In Ethereum, there is the concept of gas in order to quantify the associated costs of each transaction. The price of gas is paid using Ethereum's native currency, Ether, and we can refer to the average daily gas price as an estimate of the associated costs. In Appendix A, we see that each operation in a smart contract has a fixed gas cost, proportional to the computational complexity for that operation. For instance, every transaction has a fixed base cost of 21000 gas while adding two variables requires 3 gas. When evaluating the costs associated with the on-chain operations of our content smart contract, we are mainly interested in the following metrics:

- Transaction cost: This is based on the overall gas cost of sending data to the blockchain, and is typically consists for the following components:
    1) Base cost of a transaction
    2) Cost of a contract deployment
    3) Cost of every zero byte of data or code in a transaction
    4) Cost of every non-zero byte of data or code in a transaction [9];
- Execution cost: This indicates the portion of gas that is actually spent on executing the code in a transaction by the Ethereum Virtual Machine [13];

We have implemented the content smart contract defined in Listing 1 and deployed it to the Kovan Ethereum Test Network [12] in order to evaluate the associated costs. When deploying an instance of the content smart contract, the transaction cost is 682862 gas while the execution cost is 464038 gas. Invoking the PayForStream function costs 65800 gas in transaction costs and 44528 gas in execution costs. As on December 23rd, 2020, the average gas price is 108 gWei [14] and the price of 1 Ether is $568.41 [15]. In other words, the total cost for deploying each smart contract is about $70.4, whereas the cost of invoking PayForStream each time is about $7. However, as the price of Ether fluctuates drastically, this can be change the transaction costs greatly as well. Therefore, in future work, we hope to explore alternate blockchains, and even custom private ledgers, so that the the associated costs can be lower and more stable.

VIII. Conclusion

In this paper, we have proposed a robust incentivization and stakeholder-protection scheme named Proof-of-Stream. This scheme is modular enough to be implemented and integrated into any hybrid VoD system. We then conducted analysis and experiments to evaluate Proof-of-Stream, where the results indicate that the design goals were met. In the future, we hope to explore on-chain payments as well as alternate blockchain designs such that more stable and lower transaction and execution costs can be achieved.

References

[1] Cisco *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*

[2] K. Lee, L. Yan, A. Parekh and K. Ramchandran *A VoD System for Massively Scaled, Heterogeneous Environments: Design and Implementation*. IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2013), San Francisco, CA, August, 2013.

[3] K. Lee, H. Zhang, Z. Shao, M. Chen, A. Parekh and K. Ramchandran *An Optimized Distributed Video-on-Demand Streaming System: Theory and Design*. The 50th Allerton Conference on Communication, Control and Computing, Monticello, IL, October, 2012.

[4] H. Zhang and K. Ramchandran *A Reliable Decentralized Peer-to- Peer Video-on-Demand System Using Helpers*. Picture Coding Symposium (PCS), May, 2009.

[5] H. Zhang, M. Chen and K. Ramchandran *Scaling P2P Content Delivery Systems Reliably by Exploiting Unreliable System Resources*. IEEE MMTC E-Letter of December, 2009.

[6] H. Zhang, J. Wang, M. Chen and K. Ramchandran *Scaling Peer-to-Peer Video-on-Demand Systems Using Helpers*. IEEE International Conference on Image Processing (ICIP), Nov, 2009.

[7] S. Nakamoto *Bitcoin: A Peer-to-Peer Electronic Cash System*

[8] V. Butherin *A Next Generation Smart Contract & Decentralized Application Platform*

[9] G. Wood *Ethereum: A secure decentralised generalised transaction ledger,* Ethereum project yellow paper, vol. 151, pp. 1–32, 2014.

[10] H. Guo, E. Meamari, and C. Shen *Blockchain-inspired event recording system for autonomous vehicles* in Proc. of 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), pp. 218–222, 2018.

[11] H. Guo, W. Li, M. Nejad, and C. Shen *Access control for electronic health records with hybrid blockchain-edge architecture* arXiv preprint arXiv:1906.01188, 2019.

[12] Kovan etherum test net. [Online]. Available: https://kovan. etherscan.io

[13] H. Duan, Y. Zheng, Y. Du, A. Zhou, C. Wang, and M. H. Au, *Aggregating crowd wisdom via blockchain: A private, correct, and robust realization,* Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom2019), pp. 43–52, 2019.

[14] https://etherscan.io/gastracker

[15] https://www.coindesk.com/price/ether

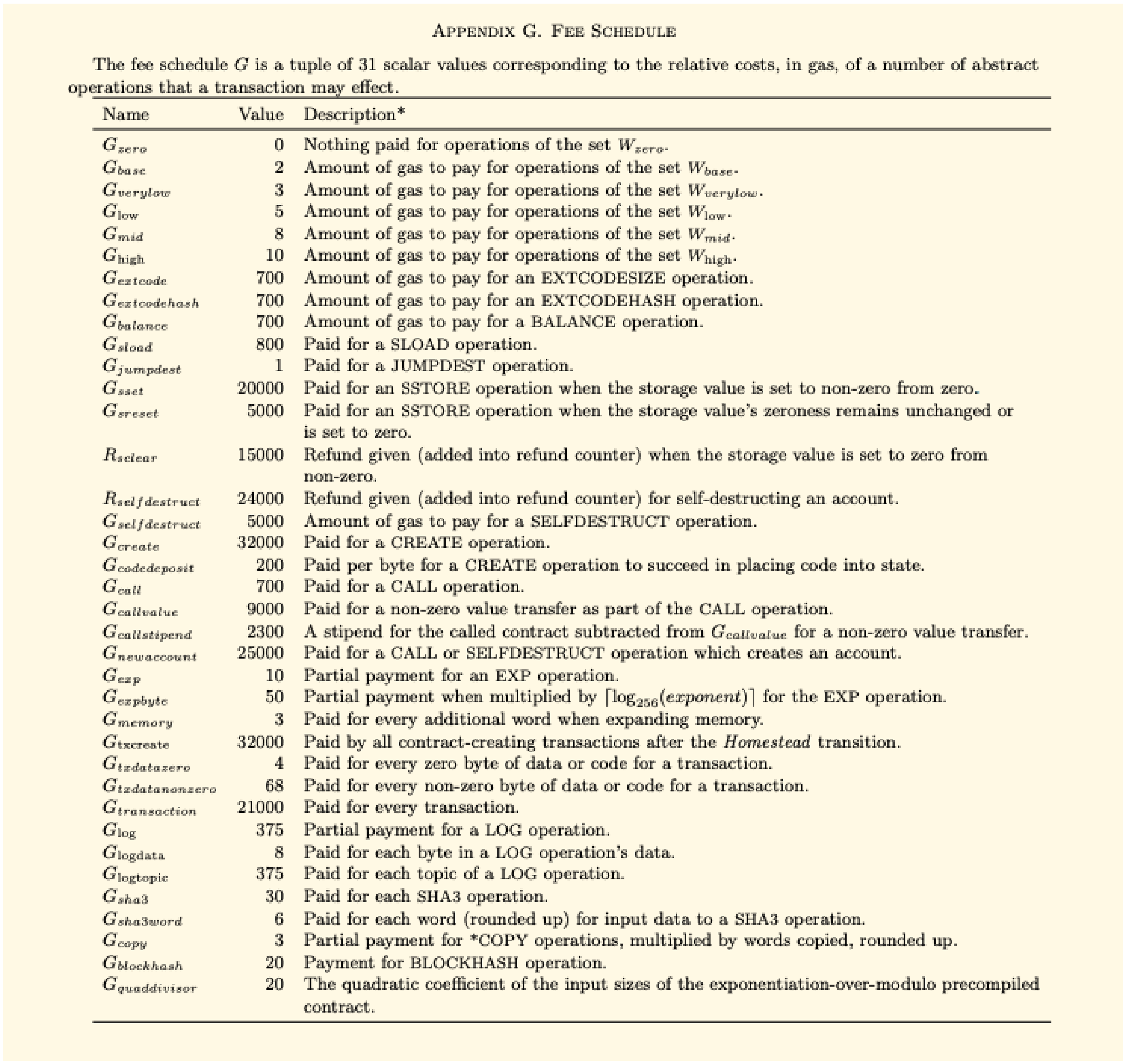
[16] S. K. Sathish, A. A. Patankar and H. Khanna, *Aurum: A blockchain based decentralized video streaming platform* 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 2019, pp. 1-8, doi: 10.1109/WCNC.2019.8886050.

[17] *MyTVChain* https://mytvchain.io/MyTVchain-wp-EN.pdf

[18] D. Petkanics, E. Tang *Livepeer: Protocol and Economic Incentives For a Decentralized Live Video Streaming Network* https://github.com/livepeer/wiki/blob/master/WHITEPAPER.md

Appendix

A. Ethereum Gas Cost



Appendix G. Fee Schedule

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for an EXTCODESIZE operation.
$G_{extcodehash}$	700	Amount of gas to pay for an EXTCODEHASH operation.
$G_{balance}$	700	Amount of gas to pay for a BALANCE operation.
G_{sload}	800	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	50	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
G_{txcreate}	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
G_{logdata}	8	Paid for each byte in a LOG operation's data.
G_{logtopic}	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.
$G_{quaddivisor}$	20	The quadratic coefficient of the input sizes of the exponentiation-over-modulo precompiled contract.