applied sciences

MDPI

Article

Rentable CDN Using Blockchain and Proof of Provenance

Suah Kim [1,*], Vasily Sachnev [2] and Hyoung Joong Kim [1]

[1] Department of Information Security, Institute of Cyber Security & Privacy, School of Cybersecurity, Korea University, Seoul 02841, Korea; khj-@korea.ac.kr

[2] Department of Information, Communications and Electronic Engineering, Catholic University of Korea, Bucheon 14662, Korea; bassvasys@hotmail.com

* Correspondence: suahnkim@gmail.com or suahnkim@korea.ac.kr

Received: 15 July 2020; Accepted: 18 September 2020; Published: 20 September 2020

Featured Application: Rentable content delivery network with proof of provenance provides s p2p based content distribution network infrastructure that supports content control, is less vulnerable to variety of collusion attacks, and provides faster content transfer speed.

Abstract: Building a rentable content delivery network (CDN), a p2p based CDN that supports content control and is composed of hardware rented from consumers, requires significant trust between the renters and consumers. The proposed method solves this using a Blockchain to provide transparency in running the network, and proof of provenance, a verifiable proof origin of the content that is used to determine who can get paid for their services to discourage malicious activities.

Keywords: blockchain; content delivery network; proof of provenance; rentable CDN; p2p based CDN

---

1. Introduction

Internet is an ever-growing network infrastructure that connects users from all around the world. With the growing adoption of personal computers and various mobile devices that connect to it, the number of accessible contents has increased as well. This meant that the sudden spike in request for a particular content has become a common occurrence and services can stop suddenly due to high load [1]. To make the contents accessible even during the sudden peak period, content delivery network (CDN) is developed. CDN solves this issue by replicating contents and redirecting content requests to ensure that the service can function uninterrupted.

CDN's performance boils down to its replication and redirecting capability [2]. For effective content replication, following three questions need to be answered: which contents should be replicated, how much replications are required, and where to place the CDN servers that will be replicating and delivering the contents.

Determining which content to replicate can be further broken down to whether to replicate the content fully or partially. This is because a content is usually composed of several files. For example, a website (content) may contain HTML files, image files, and functional codes such as JavaScript. Replicating parts that are unique to the content such as HTML files will have smaller benefit, whereas images or functional codes that are common to other contents will have bigger benefit. The identification of such files can be achieved by clustering individual files based on a metric such as the of time delay reduction measurement to delivery contents [3,4]. Then, replication priority per cluster can be determined using the same or similar metric.

On the other hand, finding the optimal level of replications (how much to replicate) is known to be an NP-hard problem [5]. Instead of trying to find the optimal level of replications, heuristic methods such as global greedy algorithm are used to find the sub optimal solution. Refs. [6,7] focused on finding a set of replication levels per object that guarantees a certain delivery performance. More advanced work incorporates other CDN specific data such as location of CDN servers, and distance between CDN servers to determine the level of replications [8,9].

Finally, there is a question of where to place the CDN servers. This question can be reformulated to solving the optimization problems like minimum k-median problem [10], k-HST problem [11], and set-cover problem [12]: the placement of CDN is optimized with respect to the cost function, where the cost function uses inputs such as set of possible locations for CDN, workload, latency, and other information derived from CDN.

Using the aforementioned work, one can build a fast, reliable and scalable CDN. However, building and managing a globally connected CDN infrastructure is costly and difficult. To remove the need for individual businesses to build and maintain a global CDN, cloud-based CDN such as Google Cloud and Amazon CloudFront have emerged to provide a fast and scalable infrastructure that can be configured and bought over online [13]. In addition to the ease of usage, cloud based CDN hardware is regularly upgraded and maintained by the CDN service provider. Despite the advantages, CDN services are costly and the inherent high barrier to enter the CDN market promotes centralization, censorship, and market oligopoly.

To reduce the cost of building and maintaining the CDN, there has been work that utilize user's existing hardware to build a globally connected CDN. Unlike the traditional CDN that uses server client model, where dedicated CDN servers deliver contents to users, this type of CDN uses peer to peer (p2p) communication to deliver contents from a set of users to a user. This is referred as p2p based CDN.

The key feature of p2p based CDN is that users can deliver contents using the resources they already own. The communication within the network is passed from a peer to a peer, thus it is designed to allow anyone with the hardware to participate, making the network easily scalable. It also means that searching requires more efforts than server client model-based networks because there is no centralized server that keeps the locations of the contents.

The most obvious method to search a content in p2p network is called flooding. This is where the search request is sent out to all the connected peers and the requests are relayed to their peers until the content is found [14]. More advanced version of flooding improves upon how the contents and peers are indexed. Improvements in content indexing reduces the search space while the improvements in peer indexing allows for more efficient way to keep track of which peers have which contents. The most famous one is called Kademlia DHT system. In this system, pieces of contents and the peer associated with having that piece is stored as a keyword and a string which resembles the hash of the keyword, respectively [15]. Thus, searching for content becomes faster and the network overhead is reduced greatly.

Although it has many benefits, p2p based CDN lacks many global features, such as caching, content moderation, load balancing, financial incentive, and reliability. Although lack of reliability is a serious issue, since users can shutdown their hardware at any time and bring down the CDN, more important issue is that there is no payment system that can financially reward users for their content delivery efforts. Without the financial incentive, users may not have good reasons to participate in p2p based CDN for prolong period.

With the rise in the development of technologies related to Blockchain, a new type of p2p based CDN became available: we refer to it as "rentable CDN". In rentable CDN, the content is delivered using p2p communications and the content delivers are paid using decentralized payment system based on Blockchain. It provides affordable localized content delivery network for small and medium-sized content distributing companies without the expensive initial investment and risks from continued operation. It also provides income for consumers with what they already have, making the market fair, and promote fair pricing. The detailed explanation of existing work on rentable CDN is expanded in the next section.

The rest of the paper is described using existing Blockchain terminology and p2p protocols to aid with understanding, but the proposed method is not bound by a specific Blockchain or p2p software.

2. Limitations of Rentable CDNs

The main problem for rentable CDN is finding a secure and trustable payment scheme based on p2p network usage. In traditional CDN, like cloud based CDNs, the user pays the CDN based on the network usage that CDN provides. On the other hand, payment is a bit trickier in p2p based CDNs like rentable CDN, because there is no third party that can verify the network usage. Figure 1 graphically shows the payment process of the traditional CDN and rentable CDN. Payment in rentable CDN is determined using the claim and the acknowledgment made by the downloader and uploader, respectively. However, "claiming" you sent a content or conversely not acknowledging that you received a content is trivial.

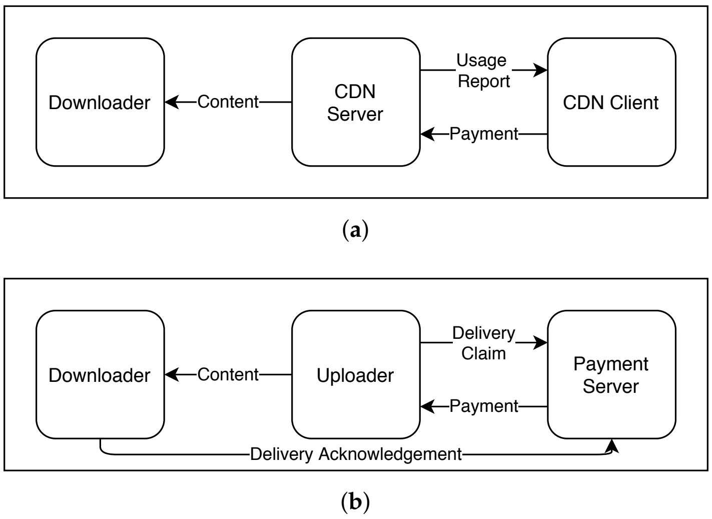

Figure 1. CDN usage based payment comparison. (**a**) charging based on CDN server network usage: content delivery cost is determined using measured usage reported by the server; (**b**) charging based on p2p network usage: content delivery cost is determined using delivery claim (Uploader) and delivery acknowledgement (Downloader).

Existing work solve this by putting constraints on the content delivery process and penalizing malicious party. For example, SLIVER.TV (see the whitepaper published in https://www.thetatoken.org/, downloadable at https://s3.us-east-2.amazonaws.com/assets.thetatoken.org/Theta-white-paper-latest.pdf?v=1597976373.104) is the most recent state of the art p2p based CDN using Theta Network (Blockchain) with multi-level BFT as the consensus mechanism. Figure 2 graphically describes how the payment is transferred. First, the downloader makes a deposit to Blockchain in the case of double spending, then they send a partially signed payment receipt to all the uploaders. When the uploader sends a part of the content to the downloader, the downloader sends another partially signed payment receipt. This is repeated until the content is downloaded fully. To get paid, the uploader can sign the partially signed payment receipt and send it to Blockchain. Note that only the first partially signed payment receipt is given freely by the downloader and additional receipts are given only to the uploader who has sent part of the content.

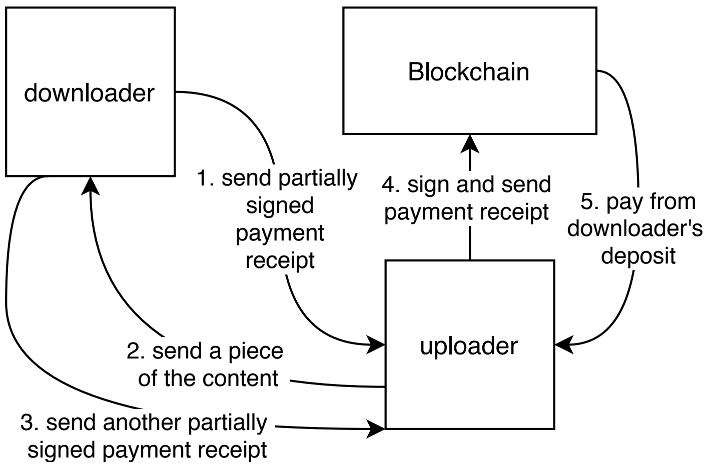
downloader
Blockchain
uploader
1. send partially signed payment receipt
2. send a piece of the content
3. send another partially signed payment receipt
4. sign and send payment receipt
5. pay from downloader's deposit

Figure 2. Payment based on p2p network usage in Theta network. The very first partially signed payment receipt is given freely, and the subsequent ones are given only if the uploader sends a piece of the content.

They explain that there are two cases for collusion. In the first case, the downloader colludes with an uploader and gives them more partially signed payment receipts than what they deserve. This double spending is easily detected as a total claim from uploaders will exceed the content's worth and the downloader pays the excess from the deposit. Thus, double spending is useless as long as the deposit is more than the sum of the false claims. In the second case, the malicious uploader does not deliver the content after receiving the partially signed payment receipts. Only the first partially signed payment receipt is sent without any conditions, and the subsequent partially signed payment receipts are sent only if the downloader receives a part of the content. Thus, the downloader is expected to lose only a little bit as payment from a single payment receipt is very small.

However, unlike their claims, their service is vulnerable to collusion attacks. Their protection against collusion and double spending is based on the fact that adversary gain is small. But what if the network is flooded with many malicious accounts controlled by one adversary? Since small loss is determined to be a normal behavior, it is impossible to identify the malicious uploader; malicious uploaders will drain the funds from the downloader (the same argument works for the malicious downloader colluding with a malicious uploader). This shortcoming is demonstrated using scalable collusion attack simulation in the later section.

In the proposed method, we take a different approach to protect against collusion. Instead of trying to identify who is malicious, we start with the assumption that all users are malicious. Then, we use the idea of proof of provenance, a simple verifiable proof that proves the origin of the content to limit who can get paid for delivering the content.

3. Proposed Method

Rentable CDN uses the idea of distributing the risks and initial investments required to run a CDN to the content creator, the content downloader, and the content uploader, while also providing infrastructure to manage specific mistrusts that content creators and content uploaders have difficulty in solving. Figure 3 shows the four roles in the proposed rentable CDN and their description is described in Table 1.

Under the proposed method, rentable CDN provides accountable content delivery service with low risk and initial investment, ideal for small and medium content distributing companies.

The next subsection discusses the detailed life cycle of the rentable CDN, such as content registration, content delivery, payment settlement, and content control.

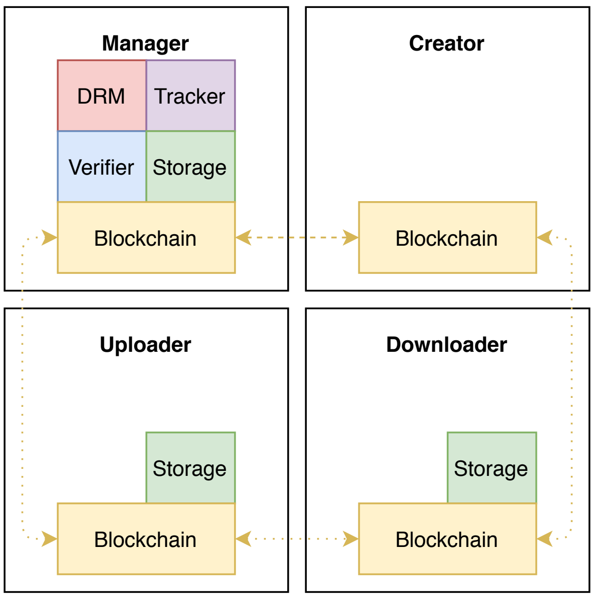

Figure 3. Four roles in Rentable CDN and their respective subsystems. Under the proposed method, it is possible to have multiple Uploaders for downloading one content.

Table 1. Description of roles in rentable CDN.

Roles	**Description**
Manager	Protects the content using **DRM** server, supports p2p communications by hosting **Tracker**, stores content in **Storage**, and verifies & signs transactions related to the content delivery using **Verifier**. Supports content control such as removing illegal contents by destroying content registration contract from **Blockchain**.
Creator	Uploads content to the **DRM** server, sets appropriate fee rates for their content, and signs and uploads the content registration contract to **Blockchain**.
Uploader	Stores the content in **Storage** and delivers them to Downloaders. Sign and send proof of piece given by a Downloader to **Verifier** to get paid.
Downloader	Uploads the content delivery contract to **Blockchain**. Checks proof of provenance of Uploader via **Blockchain**, and sends them a partially signed proof of piece for downloading pieces of the content to **Storage**. Pays appropriate fees to Manager, Creator, and Uploader using **Blockchain**.

3.1. Content Registration

Before the content is registered to the rentable CDN, Creator must upload the content to the DRM (Digital Rights Management) server. This step allows the DRM server to check if the content is legal or illegal, and either reject the registration or apply an appropriate protection to it. Instead of checking for legality, it can also support to detect and reject the registration of certain type of contents based on the network's needs. Figure 4 describes the process graphically.

The protected content is then moved to Storage, and the content registration contract that is signed by Manager is sent to Creator, so that they can sign it and uploaded it to Blockchain. The content registration contract contains information that describes the content, such as content registration contract ID, content ID, number of expected pieces, and fees. Fees refer to the content consumption fee and delivery fee. The exact breakdown of the fee is described in Section 4.1.

3.2. Content Delivery

Once the content registration contract is successfully added to Blockchain, Downloader can send the content delivery contract to Manager to be verified and signed before uploading it to Blockchain to initiate the content delivery process. The content delivery contract contains the contract id of the content registration contract, all the appropriate fees outlined in the content registration contract, and Downloader and Manager's signature. It should be noted that all fees should be paid as the content delivery contract is added to Blockchain or fee payment can be added as a function and Uploader can validate that all fees have been paid before start uploading the content to Downloader. This will ensure that all fees are automatically paid to the appropriate party after the content has been downloaded. The exact implementation is dependent on the type of Blockchain.

The content is downloaded using a p2p file transfer protocol, such as BitTorrent, with additional constraints. First, Tracker, which is maintained by Manager, checks that there is a valid download request by Downloader in Blockchain; then, Tracker can send swarm information to Downloader, which can then be used to initiate handshakes with Uploaders. The handshake initialization process is shown in Figure 5.

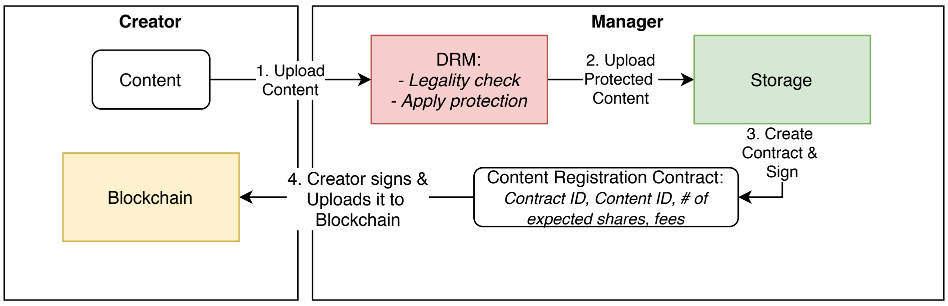
Creator
Content
1. Upload Content
Manager
DRM:
- Legality check
- Apply protection
2. Upload Protected Content
Storage
3. Create Contract & Sign
Content Registration Contract: Contract ID, Content ID, # of expected shares, fees
4. Creator signs & Uploads it to Blockchain
Blockchain

Figure 4. Content registration step.

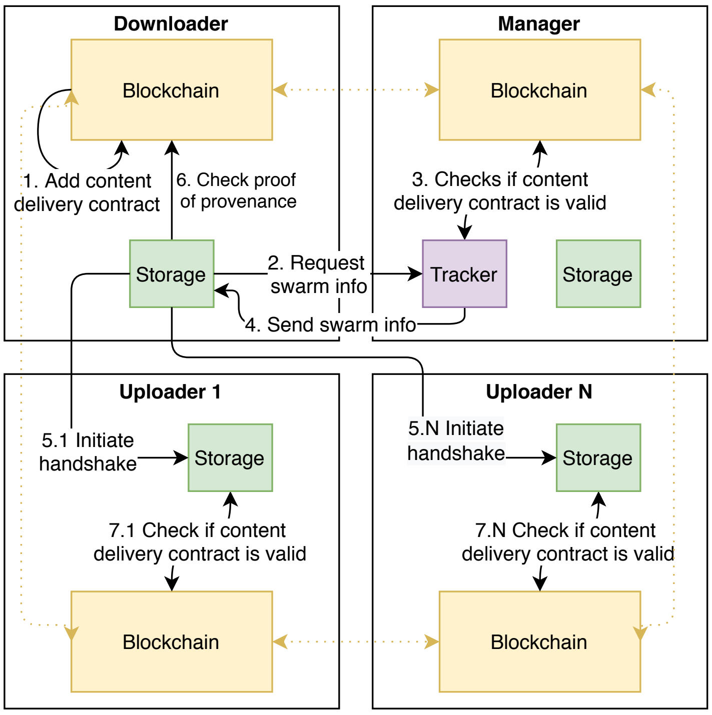
Downloader
Blockchain
1. Add content delivery contract
6. Check proof of provenance
Storage
2. Request swarm info
4. Send swarm info
Manager
Blockchain
3. Checks if content delivery contract is valid
Tracker
Storage
Uploader 1
5.1 Initiate handshake
Storage
7.1 Check if content delivery contract is valid
Blockchain
Uploader N
5.N Initiate handshake
Storage
7.N Check if content delivery contract is valid
Blockchain

Figure 5. Handshake initiation process is required to download the content.

During the handshake, Uploader sends proof of provenance for the content to Downloader and Downloader verifies it over Blockchain before requesting pieces to download.

The content delivery protocol is designed to ensure that the parties involved are paid fairly. The content is first divided into pieces and transferred using a p2p protocol. Pieces are requested using a piece identifier, such as hash, and Uploader sends an appropriate piece to Downloader.

Example of the content delivery process is shown in Figure 6. In this example, Downloader requests pieces from Uploader 1 and Uploader N. When Downloader receives a piece, it validates, sends a signed proof of piece to Uploader, and a piece received ACK (acknowledgment) as well.

The description of the proof of piece is summarized in Table 2.

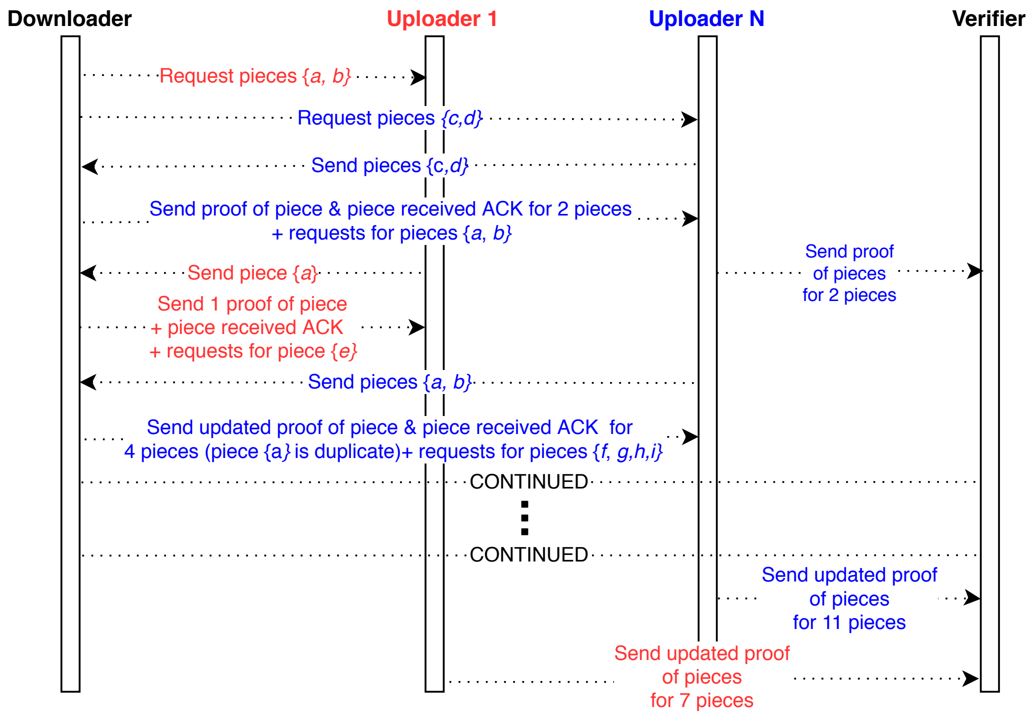
Downloader
Uploader 1
Uploader N
Verifier
Request pieces {a, b}
Request pieces {c,d}
Send pieces {c,d}
Send proof of piece & piece received ACK for 2 pieces + requests for pieces {a, b}
Send proof of pieces for 2 pieces
Send piece {a}
Send 1 proof of piece + piece received ACK + requests for piece {e}
Send pieces {a, b}
Send updated proof of piece & piece received ACK for 4 pieces (piece {a} is duplicate)+ requests for pieces {f, g,h,i}
CONTINUED
CONTINUED
Send updated proof of pieces for 11 pieces
Send updated proof of pieces for 7 pieces

Figure 6. Example of content delivery process after the handshake.

Table 2. Description of proof of piece in rentable CDN.

Parts of Proof of Piece	Description
Content delivery contract ID	Contract ID that is created when Downloader uploads the content delivery contract to Blockchain. Used as a reference to match the proof of piece with the content
Total # of new pieces	Total number of pieces that Uploader sent that Downloader did not own before.
Total # of duplicate pieces	Total number of pieces that Uploader sent that Downloader did own before.
Proof of provenance	Used by Downloader and Verifier to check that Uploader is legally allowed to deliver pieces.
Downloader's signature	Downloader signs it to prevent tampering.
Uploader's signature	Uploader signs the partially signed proof of piece sent by Downloader to confirm the content of the proof of piece.

The proof of piece contains content delivery contract id, total number of new pieces, total number of duplicate pieces, proof of provenance of Uploader for this content, and signatures of Downloader and Uploader.

When Downloader sends the signed proof of piece to Uploader, the piece received ACK is sent as well. The piece received ACK, which contains the unique piece identifier, can be used by Uploader to track which pieces have been received by Downloader. The signed proof of piece is signed again by Uploader and is sent to Verifier for verification. This bidirectionally signed proof of piece is used for payment settlement.

Under this scheme, a proof of piece is generated every time a piece is delivered. However, maintaining and verifying all pieces is taxing for Verifier. To reduce the strain, it is advised that the proof is sent not every time, but only the most updated proof is sent in a scheduled manner.

Slow Start Algorithm

Receiving a duplicate piece, a piece that has been received before, is expected to occur in p2p network. If duplicate pieces are not paid for, the content transfer speed is expected to slow down and delay the payment to Uploaders. On the other hand, if a malicious Uploader only sends duplicate pieces, this is also a problem. Thus, a good solution is for Downloader to never create a proof of piece with the number of duplicate pieces greater than the number of new pieces.

To reduce the number of duplicate pieces, it makes sense to use the slow start algorithm. The proposed slow start algorithm begins with non-overlapping piece requests to Uploader, and Uploader sends new pieces only when they received the proof of piece. Since multiple requests cannot be made, the transfer speed is expected to be slow. But, after a threshold of number of new pieces is received, multiple pieces can be requested at once to greatly increase the speed.

Slow start is used to build trust between Downloader and Uploader, and in a short time, the transfer speed can be greatly increased without being concerned about duplicate pieces.

3.3. Payment Settlement

Verifier always keeps the most updated proof of piece (proof with the highest combined number of "Total # of new pieces" and "Total # of duplicated pieces") that has been received from each Uploader, and then starts a payment settling process when Downloader claims that the content has been finished downloading or if the expected number of proof of pieces (new pieces) are received.

Verifier checks if Downloader correctly downloaded the content using an interactive challenge and response (Verifier uses the content stored in Manager's Storage to verify the response). There are three possible responses: (1) correct response, which means that Downloader downloaded the content correctly, and the proof of provenance is generated, (2) incorrect response, which means Downloader needs to download more or correct pieces, (3) a contract termination response, which means that Downloader wants to stop downloading and wants a refund.

The payment settling function, which is a function of the content delivery contract, is used to trigger a batched payment to appropriate parties. However, the payment settlement transaction is not validated over Blockchain, rather it is uploaded as an invoice record used to pay the involved parties. Only Verifier verifies the information offline to reduce the computation on Blockchain.

In the case that Downloader requests to terminate the contract or claim that download has been finished, Manager sends terminating notice to Uploaders, which stops the uploading of the content and the most updated proof of piece is queried by Verifier. This step ensures that all Uploaders are paid fully even if the contract becomes terminated.

Proof of Provenance

The main concern for decentralized CDN is the trust between Downloader and Uploader. It is very easy for Uploader to claim that they uploaded more than they did. Conversely, Downloader can claim that they did not download from a specific Uploader. So, how can we use provenance to solve this?

Provenance is widely used to assist with authenticating objects, such as artworks and wines, by comparing it with a historic record. It also is used in crime investigation to track evidence through a chain of commands. Like the examples, the proposed proof of provenance tracks the origin of the content and can prove that the content has been legally obtained from rentable CDN.

The structure of the proof of provenance is described in Table 3. It contains content delivery contract ID, provenance flag, Uploader's address, and Verifier's signature. As mentioned before, Uploader's proof of provenance must be sent to Downloader during the handshake for verification.

Table 3. Description of proof of provenance in rentable CDN.

Parts of Proof of Provenance	Description
Content delivery contract ID	Contract ID in Blockchain that holds the proof of provenance.
Provenance flag	This is a binary flag, where '1' indicates that the provenance exists and '0' indicates that the content delivery contract has been terminated without downloading the whole content.
Uploader's address	Specifies which Uploader is the proof of provenance valid for.
Verifier's signature	Verifier's signature which confirms that proof of provenance is genuine.

Every time the content is finished downloading, Verifier issues a challenge and response query to Downloader to check if they correctly downloaded the whole content. If they respond correctly, then a proof of provenance is generated and is added as a part of payment settling transaction. This proof is then used as a reference for any future content delivery process to prove the provenance of the content.

Figure 7 shows how the proof of provenance is distributed. In the beginning, only Manager can deliver the content, because no one else has the content. The content registration contract ID is substituted as content delivery contract ID for the proof of provenance because that is how Manager acquired the content. Naturally, when Downloader finishes downloading and successfully has a record of proof of provenance on Blockchain, they can become Uploader and deliver contents to other Downloader(s).

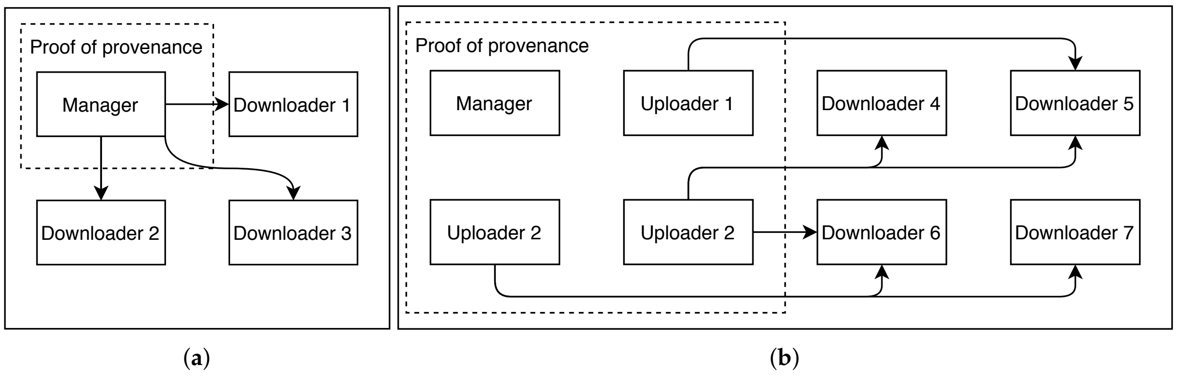

Figure 7. Distribution of Proof of provenance. (**a**) Only Manager can upload in the beginning. (**b**) Downloaders 1, 2, and 3 have acquired proof of provenance and became Uploaders 1, 2, and 3, and deliver the content to other Downloaders.

During the payment settlement, if there is an incorrect proof of provenance (or cannot be verified) in the proof of piece, that Uploader is not eligible for the payment and Downloader is not eligible for proof of provenance for this content delivery contract. Furthermore, any unclaimed payment including the case of incorrect proof of provenance is paid to Verifier to prevent Downloader from cheating.

3.4. Content Control

Sometimes a content must be deleted because it violates a rule set by the CDN, such as a legal reason, or Downloader may decide to stop downloading the content and request a refund. For the former case, Manager should destroy the content registration contract so that no new download contracts can be created. In the case where Downloader decides to stop downloading, they can request to terminate their content delivery contract via triggering the payment settlement process.

4. Analysis

This section provides an analysis of the proposed scheme with regards to the fee structure, and possible collusion attacks.

4.1. Fee Structure

The proposed fee structure exists to discourage malicious activities, such as colluding and cheating while encouraging fast transfer speeds and fair pricing.

There are largely two fees: content consumption fee and content delivery fee. Content consumption fee is set by Creator, whereas the content delivery fee is a fixed amount relative to the content size. Content delivery fee can be further broken down to verification fee, delivery fee, network maintenance fee, and delivery deposit fee. The verification fee is paid to Verifier for verifying proof of pieces. The delivery fee is paid to Uploader for delivering the content to Downloader. The network maintenance fee is used as an incentive for Uploaders to keep the content. Finally, the deposit fee, which is equal to the verification and transfer fee is allocated as well. This deposit is used to pay for the duplicate pieces and to discourage collusion and cheating. Any leftover deposit is returned to Downloader during the payment settling phase.

Table 4 shows an example breakdown of the content delivery fee. To discourage collusion, Verification fee should be strictly greater than the delivery fee.

Table 4. Example of the content delivery fee breakdown for X mb sized content. c_1, c_2, c_3, and c_4 are some constants defined by Manager.

Content Delivery Fee	Content Size	Piece Size	Piece #	Price/mb	Price/Piece
Verification fee				0	c_1
Transfer fee	X mb	0.256 mb	$\frac{X}{0.256}$	c_2	c_3
Network Maintenance fee				c_4	0

The piece size is set to 0.256 mb, as this is recommended size for BitTorrent, but this can be modified as needed. The verification fee is only dependent on the number of pieces because the verification is done per proof of piece. The delivery fee is dependent on both the size and the number of the pieces since network usage is based on the content size and the number of delivered pieces. Unlike the other two fees, network maintenance fee is only dependent on the size, but this is equally shared among all Uploaders who sent in proof of pieces for that content and Verifier. This fee is provided to Uploaders for hosting the content in Storage, and Verifier for maintaining infrastructure for its verification service.

For a simple example, suppose Uploader A and Uploader B have delivered a content of sized 256 mb (1000 pieces) to Downloader A, and $c_1 = 0.03, c_2 = 0.01, c_3 = 0.01, c_4 = 0.5$. Then, content download contract has $2 \times 1000 \times 0.03 = 60$ for verification fee and the deposit, $2 \times 256 \times 0.01 + 2 \times 1000 \times 0.01 = 25.12$ for delivery fee and the deposit, and $256 \times 0.05 = 12.8$ for the network maintenance fee. Then, suppose Uploader A delivered 500 new pieces and 40 duplicate pieces, and Uploader B delivered 500 new pieces and 2 duplicate pieces. Then, Verifier receives $1042 \times 0.03 = 31.26$ for the verification fee, Uploader A receives $540 \times 0.256 \times 0.01 + 540 \times 0.01 = 6.7824$ for the delivery fee, and Uploader B receives $502 \times 0.256 \times 0.01 + 502 \times 0.01 = 6.30512$ for the delivery fee. Finally, network maintenance fee of $0.05 \times 256 = 12.8$ is shared evenly between Uploaders and Verifier. In total, contract held 97.92 and fee of 57.14752 was used, thus 40.77248 is returned to Downloader A.

4.2. Collusion Attacks

Collusion attack is the main concern in rentable CDN. Collusion is possible because it is very difficult to prove to a third party that a part of the content has been transferred or has been received.

Collusion attacks can be initiated either by the malicious Uploader or Downloader. In the case of the malicious Uploader, their goal is to be paid without uploading any contents. In the case of the malicious Downloader, their goal is to avoid paying the Uploader.

The most recent commercialized version of rentable CDN called Theta network is vulnerable to collusion attack. Their shortcoming is from allowing collusion by malicious Uploaders. They argue that even though a malicious Uploader can cheat by claiming payment for one piece without delivering the first piece, the overall loss for the Downloader is negligible because the payment for a single piece is small. However, we will prove that this claim is not true in the next subsection by demonstrating a scalable collusion attack.

4.3. Scalable Collusion Attack Simulations

As explained in the introduction, the proposed rentable CDN is not bound to a specific Blockchain or p2p software. The reference software is built using Ethereum as a mainchain, and another Blockchain as a sidechain which supports delegated proof of stake. Ethereum is used to secure the interactions between Ethereum and the side chain, while the side chain is used to support all necessary transactions to run the rentable CDN. For p2p communication, Bittorent protocol is simulated.

To measure how the proposed method improves upon the exiting work, collusion simulations are conducted. In the simulations, we measure the difference in download speeds and Downloader's loss due to collusion. Since the performance can vary depending on hardware, peers, location, and file availability, we try to simulate an environment with an average network connection.

Figure 8 shows the simulation result when 700 MB file is downloaded with 50 honest Uploaders and 10 malicious Uploaders. Theta network and the proposed method are compared in terms of how long it takes to download the file. "Loss due to collusion" represents the loss in payment (relative to file size) for Downloader due to collusion in Theta network; the proposed method does not have any loss due to collusion because of the proof of provenance system. The graph clearly shows that the proposed method has a slightly faster download speed then Theta network. This not a surprising result because the proposed method's download speed is not affected by malicious Uploaders. The close view of the figure also shows that "loss due to collusion" only goes up to 0.37%. This is a negligible loss as Theta network has claimed, however, we also observe that the maximum loss due to collusion occurs within the first 5 seconds of the connection. This means that the malicious Uploader only needs to connect for 5 seconds and then attack another Downloader. This inherent structure allows a single malicious Uploader to scale their attack to the whole network.

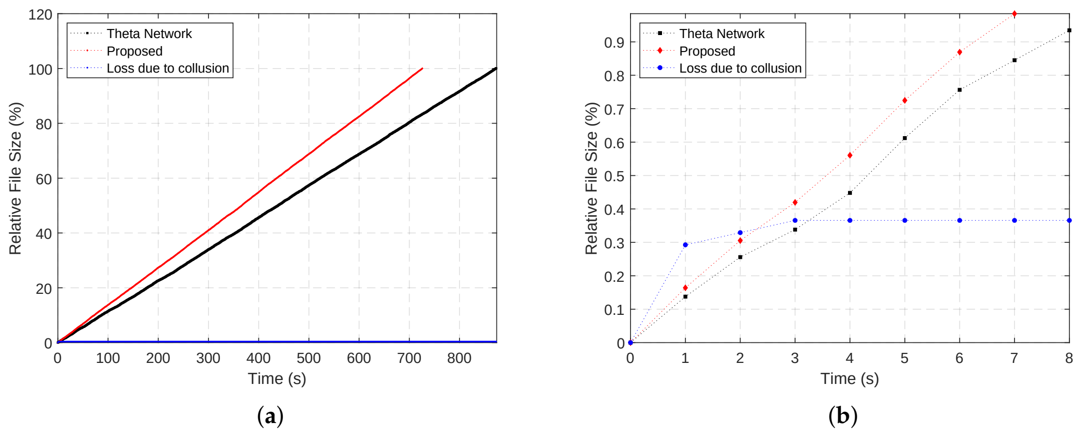

Figure 8. Download speed comparison between Theta network and proposed method. (**a**) 50 honest Uploaders and 10 malicious Uploaders. (**b**) close-up view of the left figure.

Figure 9 shows extended simulation results when collusion attack is scaled, i.e., overwhelming number of fake malicious Uploaders disrupt the network. In the simulation, there are 6 and 12 times more malicious Uploaders than the honest Uploaders. The figure clearly shows that contents take longer time to download in Theta network than the proposed method, while the loss due to the collusion increases to a significant amount. The content takes longer time to download for Theta network because malicious Uploaders take up the download connection spots that honest Uploaders could have connected to.

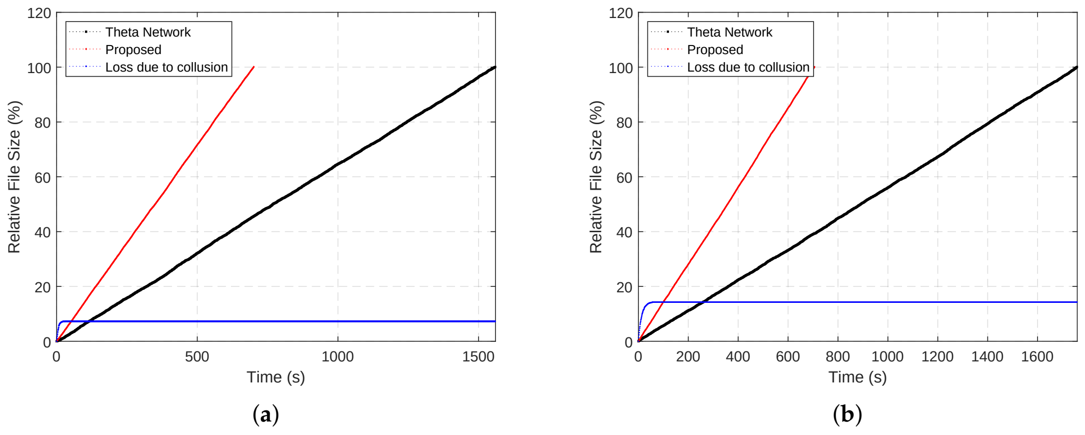

Figure 9. Download speed comparison between the proposed and Theta network when malicious peers (300 and 600) overtake the network. (**a**) 50 honest Uploaders and 300 malicious Uploaders; (**b**) 50 honest Uploaders and 600 malicious Uploaders.

4.4. Security Analysis for the Proposed Method

In rentable CDN, Uploader should not be able to reuse a proof of piece that has been claimed already, and Downloader should not be allowed to redirect payment for a legitimate Uploader to themselves or Uploaders that they are colluding with. The former is called "replay attack" and the latter is called "collusion attack". The proposed method is secure against replay attacks using proof of pieces, and various collusion attacks.

Suppose a malicious Uploader wants to replay a previously used proof of piece to get paid again. Consider the following possible replay attack scenarios:

1. *Proof of piece to be replayed was already used by another Uploader:* it cannot be replayed because the proof of piece contains "Uploader's address" and "Downloader's signature". The malicious Uploader cannot replace existing Uploader's address with their own address since the signature will not match with the signed data.
2. *Proof of piece to be replayed was already used by themselves:* it cannot be replayed because the proof of piece contains "Content delivery contract ID", "Downloader's signature". "Content delivery contract ID" ensures that the proof of piece is only valid for this current contract and cannot be replayed for a different contract. Also, replaying the piece again does not increase the payment unless "Total # of new pieces", and "Total # of duplicate pieces" are modified, but this is not possible since Downloader's signature will not match the modified data.

Suppose a malicious user want to collude to get paid for the service they did not provide. Consider the following collusion attack scenarios:

1. *Malicious Downloader colludes with fake Uploaders:* Suppose a malicious Downloader creates fake Uploaders and claim that content is downloaded from them. Under the proposed scheme, fake Uploaders are not paid without the correct proof of provenance, and malicious Downloader will not be able to become an Uploader for the content if they send a proof of piece with false proof of provenance. Furthermore, Downloader must send correct proof of piece to the honest Uploaders, otherwise, Uploaders will not send additional pieces beyond the first piece.
2. *Malicious Downloader colludes with other Uploaders:* Suppose a malicious Downloader claim that content has been downloaded from the colluders. Under the proposed fee structure, Downloader does not gain from this because honest Uploaders still need to be paid. If anything, their loss will grow as additional verification and network maintenance fees have to be paid for the proof of pieces submitted by the colluders.

3. *Malicious Uploader:* In this scenario, a malicious Uploader tries to gain a legitimate proof of piece from Downloader without sending a piece of the content. Unlike the Theta network where the payment is given before Uploader sends a piece of the content, the proposed method only sends the proof of piece if a piece is delivered. In other words, malicious Uploaders cannot gain from colluding.

The combination of proof of piece, proof of provenance, and fee structure which penalizes malicious activities, offers protections against replay attacks using proof of piece and collusion attacks.

5. Conclusions

Consumers are downloading and uploading contents more than ever, but there are almost no small or medium scale content providers due to the high initial risk associated with developing a content distribution network (CDN).

Rentable CDN provides an alternative way to distribute contents then the traditional CDN and decentralized CDN. It uses Blockchain to decentralize the payment process and p2p communication for transferring the content between the users. Since the payment is decentralized, all payment operations are transparent. The contents are transferred using the hardware already owned by users to reduce the initial hardware investment.

Collusion attack is one of the main concerns when operating a p2p based CDN. This is due to lack of robust verification method by a third party to check whether a content has been transferred from a user to another user. Thus, malicious users can use variety of collusion attacks to get paid for the services that they did not provide and reduce the content transfer speed. Existing work tries to make collusion attacks difficult by putting constraints on how content is transferred and penalizing malicious users to discourage collusion. Even though the loss due to collusion per content is small, the attacks are easily scalable and reduce the content transfer speed significantly. To demonstrate the scalable collusion attack, we simulate the collusion attack and compare the existing work with the proposed method. The simulation results show that Downloaders download the content faster, and do not incur loss due to collusion when the proposed method is used.

The proposed method protects against collusion attacks using the proof of provenance. Proof of provenance tracks which Uploaders can upload specific contents, and penalize Uploaders and Downloaders based on that information. Compare to the existing work, the proposed method provides faster download speed and is not vulnerable against scalable collusion attacks. It is also secure against replay attacks that try to claim previously claimed proof of piece to get paid again without doing the work. This is achieved by designing proof of piece to be unique with respect to the content contract.

To conclude, the proposed method provides faster download speed than existing work while being secure against scalable collusion attacks that existing p2p based CDNs are vulnerable against.

Author Contributions: Conceptualization, S.K., V.S., and H.J.K.; methodology, S.K.; software, S.K.; validation, S.K., V.S., and H.J.K.; formal analysis, S.K.; investigation, S.K.; resources, S.K.; data curation, S.K., V.S., and H.J.K.; writing—original draft preparation, S.K.; writing—review and editing, S.K. and V.S.; visualization, S.K.; supervision, V.S. and H.J.K.; project administration, S.K.; funding acquisition, S.K. and H.J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by an Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government (MSIP) (No.2018-0-00365, Development of on-off hybrid blockchain technology for real-time large-scale data distribution).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jung, J.; Krishnamurthy, B.; Rabinovich, M. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In Proceedings of the 11th international conference on World Wide Web, Honolulu, HI, USA, 7–11 May 2002; pp. 293–304.
2. Wang, L.; Pai, V.; Peterson, L. The effectiveness of request redirection on CDN robustness. *ACM SIGOPS Oper. Syst. Rev.* **2002**, *36*, 345–360. [CrossRef]
3. Chen, Y.; Qiu, L.; Chen, W.; Nguyen, L.; Katz, R.H. Efficient and adaptive Web replication using content clustering. *IEEE J. Sel. Areas Commun.* **2003**, *21*, 979–994. [CrossRef]
4. Fujita, N.; Ishikawa, Y.; Iwata, A.; Izmailov, R. Coarse-grain replica management strategies for dynamic replication of Web contents. *Comput. Netw.* **2004**, *45*, 19–34. [CrossRef]
5. Chen, L.C.; Choi, H.A. Approximation Algorithms for Data Distribution with Load Balancing of Web Servers. In Proceedings of the 2001 IEEE International Conference on Cluster Computing, Las Vegas, NV, USA, 14–17 October 2001; Volume 1, p. 274.
6. Tang, X.; Xu, J. QoS-aware replica placement for content distribution. *IEEE Trans. Parallel Distrib. Syst.* **2005**, *16*, 921–932. [CrossRef]
7. Laoutaris, N.; Zissimopoulos, V.; Stavrakakis, I. On the optimization of storage capacity allocation for content distribution. *Comput. Netw.* **2005**, *47*, 409–428. [CrossRef]
8. Bektas, T.; Oguz, O.; Ouveysi, I. Designing cost-effective content distribution networks. *Comput. Oper. Res.* **2007**, *34*, 2436–2449. [CrossRef]
9. Bektaş, T.; Cordeau, J.F.; Erkut, E.; Laporte, G. Exact algorithms for the joint object placement and request routing problem in content distribution networks. *Comput. Oper. Res.* **2008**, *35*, 3860–3884. [CrossRef]
10. Mirchandani, P.B.; Francis, R.L. *Discrete Location Theory*; Wiley: New York, NY, USA, 1990.
11. Bartal, Y. Probabilistic approximation of metric spaces and its algorithmic applications. In Proceedings of the 37th Conference on Foundations of Computer Science, Burlington, VT, USA, 14–16 October 1996; pp. 184–193.
12. Karlsson, M.; Karamanolis, C. Choosing replica placement heuristics for wide-area systems. In Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan, 26 March 2004; pp. 350–359.
13. Wang, M.; Jayaraman, P.P.; Ranjan, R.; Mitra, K.; Zhang, M.; Li, E.; Khan, S.; Pathan, M.; Georgeakopoulos, D. An overview of cloud based content delivery networks: research dimensions and state-of-the-art. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 131–158.
14. Lv, Q.; Cao, P.; Cohen, E.; Li, K.; Shenker, S. Search and replication in unstructured peer-to-peer networks. In Proceedings of the 16th international conference on Supercomputing, New York, NY, USA, 22–26 June 2002; pp. 84–95.
15. Maymounkov, P.; Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 53–65.