Efficient Integrity Protection for P2P Streaming

Lingli Deng, Ziyao Xu, Wei Chen, Lu Lu, Xiaodong Duan

Efficient Integrity Protection for P2P Streaming

Lingli Deng[1]**, Ziyao Xu[2], Wei Chen[1], Lu Lu[1], and Xiaodong Duan[1]

[1] Network Department of China Mobile Research Institute,
32 Xuanwumenxi Ave, Beijing, 100053, China
denglingli,chenweiyj,lulu,duanxiaodong@chinamobile.com
[2] Alibaba Corp.,
12 Xidawang Street, Chaoyang, Beijing, 100022, China
ziyao.xu@alibaba-inc.com

Abstract. This paper proposes an efficient checksum consolidation method, where the content source publishes only a few consolidated checksums (as the direct verification proof) for the media content. A tree-based and a chain-based scheme are proposed to consolidate chunk checksums respectively for VoD and live streaming application scenarios.

Keywords: hash, chunk, integrity, P2P streaming

1 Introduction

In a typical P2P streaming system[1], there exist several well-known portals for the content sources to publish their local media content to potential audience accessing the network through the other peers. For a published media program, its responsible tracker server as well as other human readable description information, are returned to a querying peer by the portal, as the response to the latter's program selection request. The responsible tracker holds the IP/port address list for correspondent swarm (i.e. peer group uploading/downloading the same program) members. By submitting a swarm joining request to the responsible tracker, a downloading peer receives the peer list (containing sharing peers' IP/Port addresses, etc.) and hands in its own IP address and Port number to the tracker to update the swarm's peer list. From this point, a downloading peer can initiate media downloading request to other intermediate peers in the list.

To exploit the many-to-many sharing pattern of the P2P protocol, a large media file is divided into a group of chunks and each chunk is distributed independently afterwards [2]. In other words, a downloading peer needs only a single chunk, rather than the complete chunk set for the original media file, to become a server for uploading the local chunk to other peers. It is desirable from the respects of both the service provider and the downloading peers, to ensure that the media content is exactly the same as published and not manipulated by any intermediate party, especially when the provider holds certain reputation/authority/responsibility for the information's authenticity/validity it delivers to the public.

** This work is supported by Chinese National Grant 2012ZX03002008.

However, previous P2P streaming systems put little concern on integrity protection. Moreover, particular requirements for streaming applications to achieve chunk-level integrity protection make previous solutions for P2P file-sharing networks inapplicable to streaming settings. On the contrary, the consolidated method proposed in this paper is highly applicable to P2P streaming systems, for its cost-efficiency in achieving chunk-level integrity protection for chunk sets, achieved by employing hash consolidation methods (i.e. hash tree based scheme for VoD applications and hash chain based scheme for live streaming applications). It is expected to bring down the storage and verification cost for downloading peers, the publishing cost for content sources, and the storage cost for system portal/trackers.

The rest of this paper is organized as follows: Section 2 gives a brief review on related work. Section 4 analysis integrity protection requirements for P2P Streaming applications. Section 3 introduces the hash tree and hash chain mechanisms to be utilized by our proposals in Section 5. In section 6, a preliminary analysis is conducted. Section 7 concludes.

2 Related Work

Previous P2P file-downloading systems take measures to ensure file integrity from the publisher all the way to the ultimate downloading peer. For example, BitTorrent users can publish a file's hash checksum to the system portal/tracker, to be used later by the downloading peer to verify a file's integrity after downloading the complete chunk set and rebuild the original file. However, in BitTorrent's scheme, publisher only provides the single checksum for the whole file, a downloading peer cannot locate the specific chunk(s) if only part(s) of the chunk set get manipulated, which means a single manipulated chunk would cause the searching and downloading the complete file all over again. Therefore, eMule enhances BitTorrent's scheme allowing content sources to incorporate the hash checksums for each individual chunk into the published content handle, so that chunk-level integrity verification and re-transmission are viable for the downloading peer. Another drawback of BitTorrent's scheme is that the total verification work is performed after the whole set of chunks are downloaded completely, indicating an insufferable delay to the completion of file downloading transaction, indicating devastated user experience. To conquer the problem, [3] proposed to exploite the iterative feature of the hash algorithm, in order to hide the check delay into the file downloading process, where the downloading peer uses the hash algorithm to compute a chunk's checksum and updates the file's hash vector whenever it received a new chunk. After downloading the last chunk, a peer only has to compute a single chunk's hash checksum before building the file's checksum and is able to complete integrity verification rapidly.

3 Background

A single hash tree[4] is able to protect the integrity of a large number of unsafely stored data objects, once the tree's root hash is safely stored. A leaf node is created for each data object, and each internal node is the hash value of the concatenation of its direct children. It is computationally infeasible for an attacker to construct another hash tree out of some modified leaf nodes to produce the same root hash, as the internal nodes (including the root hash) are constructed with collision-free hash algorithms [5]. See Algorithm 1, where $tree(S, H, d)$ calculates the root hash r for the d-degreed tree constructed from an array S using a hash functions from H.

Algorithm 1 $tree(S, H, d)$

```
$k \leftarrow 1; s \leftarrow S.length$
while $s > 1$ do
  $O \leftarrow \emptyset; n \leftarrow \lceil s/d \rceil; temp \leftarrow S[k]$
  for $i = 1$ to $n$ do
    for $j = 1$ to $d - 1$ do
      $temp \leftarrow temp||S[k+1]; k \leftarrow k + 1$
    end for
    $O[i] \leftarrow H(temp)$
  end for
  $s \leftarrow n; S \leftarrow O$
end while
return $O[1]$
```

To check for the integrity of a leaf node, one needs to repeatedly compute the hash checksum for the parent node by concatenating the values of the node in question and its siblings and move upwards along the tree until the root is arrived. If the computed value is the same as the securely stored root hash, the leaf's integrity is intact. To update the value of a leaf node, one needs first to check the leaf's integrity (as stated above). If the value is intact, he/she can go further to modify the leaf's value, and update the leaf's parent, its grandparent, until the root hash is updated accordingly.

Intuitively, a hash chain is a hash tree, where "the height of each right-side sub-tree of any internal node is 1". Hash chains can be used to protect the integrity of a sequence.

4 Problem Statement

Definition 1 (P2P Chunk Integrity). *A chunk set has integrity, if each of its element maintains hash-integrity during the transmission chain from the content provider all the way to the downloading peer (including all the intermediate peers who downloaded the media chunk and uploaded to the next peer along the way).*

To ensure chunk-level integrity, previous P2P file-downloading systems allow a content source to provide the hash checksums for each chunk (in the form of a hash vector) as part of the descriptive information of the intended file. A downloading peer is able to verify a chunk's integrity by locally calculating its hash checksum and check the value against correspondent element in the hash vector published by the portal.

However, as summarized by Table 1, streaming applications differ from the file downloading applications in dictating the following specific requirements for chunk integrity protection.

Table 1. Special Requirements for Media Chunk Integrity Verification

Application	Real-Time Verification	Partial Publication	Off-line Verification
Live	●	●	
VoD	●		●

First of all, in streaming applications, where the downloading peer is always holding a small number of chunks (not necessarily the whole chunk set for the media program) and is required to perform chunk-level integrity verification before playing the media program locally. In other words, they dictate *real-time verification requirement* that the local verification of individual/subset of media chunks cannot rely on the program's global information (e.g. the complete chunk set or the whole vector of hash checksums).

Secondly, the non-stop feature of some 7×24 live programs makes their content sources themselves is incapable of computing and publishing the global checksum for the media beforehand statically. Indicating *partial publication requirement* that the publication of checksum information cannot rely on the program's global information.

Thirdly, it is intuitively that until the portal delete the relevant downloading link(s) from its resource sharing list out of management reasons, the system should support the efficient downloading and correct verification of media chunks of a VoD media file, once there is a live peer group who together own a complete chunk set of the file, even when the content source leaved the system temporarily or permanently. In short, VoD applications also demand *off-line verification requirement* that the local verification for individual/subset of media chunks of a VoD media program cannot rely on the program's source's presence.

5 Consolidated Integrity Verification

The heart of our proposal, is the introduction of a peer-to-peer paradigm for the hash checksum generation and distribution, for the purpose of reducing the publication and storage signaling expenditures for potentially centralized portal/tracker in ensuring chunk-level integrity protection.

The proposed method mainly involves three procedures in a P2P streaming system, including[3]: (1) *a source-portal interaction* for media and direct checksum publication, when the content source to compute the root hash and its signature locally and publishes them to the system portal; (2) *a peer-peer interaction* for indirect checksum; (3) *generation and distribution*, when the intermediate sharing peer to compute the verification paths for locally verified chunks and share them with other downloading peers; and (4) *a local verification procedure*, when the downloading peer verifies a newly downloaded chunk's integrity employing the indirect checksums (verification paths) from other swarm members as well as the direct checksums (root hashes) from the system portal.

5.1 Chain-based scheme for live streaming

The root hash of a published live channel $s(t)$, is published to the portal and updated at time t on a periodic basis (e.g. for every u chunks), by the content source using the hash chain algorithm, while the previous value $s(t-1)$ is also stored by the portal to be used by the downloading peer to verify chunk(s)'s integrity within the latest updating period (t-1,t]. The intermediate sharing peer, after downloading a media chunk x within the latest updating period, computes its hash checksum h_x and stores the value into a u-length verification vector D as $D[x]$. The verification vector $D[1..u]$ constitutes the verification path s_x for chunk x, i.e. $s_x \doteq D[1..u]$. The downloading peer needs only to acquire the verification path $D[1..u]$ from any intermediate sharing peer, as well as the root hash pair $s(t)$ and $s(t-1)$ from the portal, to verify chunk x's integrity.

Root hash's formulation and publication Assume that a content source $Peer-S$ wishes to publish a live channel X to the portal. The correspondent process of the channel's root hash's formulation and publication is as follows.

– **Step1:** $Peer-S$ locally produces the descriptive information $m(X)$ for the live channel X, and submits it, as well as the first set of u chunks x_{1u}'s root hash $s(0)$ (computed through the hash chain algorithm), to the portal for publication.

$$s(0) = chain(X[1..u], H, d) \tag{1}$$

– **Step2:** For each updating period (i.e. the time interval for playing u media chunks), $Peer-S$ locally computes the current root hash $s(t)$, and submits it to the portal for update. To respond, the portal stores $s(t)$ and $s(t-1)$ for the channel.

$$s(t) = chain(S(t-1)||X[1..u], H, d) \tag{2}$$

[3] For systems employing centralized tracker architecture, the tracker may take the responsibility of checksum publication and our proposal also applies by replacing the portal's role with tracker.

Media chunks' integrity verification Assume that $Peer-D$ wishes to downloads the xth media chunk of the latest updating period from an intermediate sharing peer $Peer-U$ and checks the chunk's integrity locally.

Two types of local verification processes are provided: an eager downloading peer may choose Process 2, while a prudent peer may choose Process 1.

1. **Local verification process 1:** $Peer-D$ checks a chunk's verification path's validity before downloading it and perform the correspondent integrity verification immediately afterwards.
   - **Step1:** $Peer-D$ checks if the local verification vector D contains x's verification path s_x: if so, executes **Step3**; or **Step2**, otherwise.
   - **Step2:** $Peer-D$ asks $Peer-U$ for verification path $s_x = D'[1..u]$, and verifies the validity of the latter's response s_x through the following equation, according to the root hash pair $s(t)$ and $s(t-1)$ from the portal:
   $$s(t) = chain(S(t-1)||D'[1..u], H, d) \tag{3}$$
   If the equation holds, the path is valid, update the local verification vector $D = s_x$ and executes **Step3**; otherwise, give up $Peer-U$ and choose another intermediate sharing peer for chunk x.
   - **Step3:** After downloading chunk x from $Peer-U$, $Peer-D$ computes its hash checksum h_x and verifies its integrity through the following equation, according to the locally stored verification path s_x:
   $$h_x = s_x[x] \tag{4}$$

   If the equation holds, chunk x is downloaded successfully; otherwise, give up $Peer-U$ and choose another intermediate sharing peer for downloading x.
2. **Local verification process 2:** $Peer-D$ performs the integrity verification after successfully downloading all the chunks of the latest updating period.
   - **Step1:** After downloading chunk x from $Peer-U$, $Peer-D$ computes its hash checksum h_x and store it to the correspondent element $D[x]$ of the verification vector D.
   - **Step2:** If all the chunks within the latest updating period are downloaded locally, execute **Step3**; otherwise, terminate the current chunk downloading transaction.
   - **Step3:** $Peer-D$ performs the integrity verification for the chunk set locally downloaded for the latest updating period, through using the local verification vector D as well as the root hash pair $s(t)$ and $s(t-1)$ from the portal, according to the following equation:
   $$s(t) = chain(S(t-1)||D[1..u], H, d) \tag{5}$$
   If the equation holds, terminate the chunk downloading transaction for the latest updating period. Otherwise, starting from the first media chunk of the current period, request the correspondent intermediate sharing peer for the chunk's verification path, until a valid path, satisfying the following equation, is received or no other peer to request:
   $$s(t) = chain(S(t-1)||D'[1..u], H, d) \tag{6}$$

If the received D' is valid, $Peer-D$ locates a set of integrity-corrupted chunks S, by comparing h_{x_i} and $D'[i]$. Otherwise, S is set of the complete set of chunks within the latest updating period.

– **Step4:** $Peer-D$ through the "local verification process 1" for relocating, downloading and verifying each chunk in set S.

Verification path's computation and distribution Assume that $Peer-U$ wishes to share its locally downloaded chunk x, and the correspondent verification path for other downloaders. Intuitively, there are two possibilities:(a) $Peer-U$ downloaded the valid verification path s_x as well as x itself from some other intermediate sharing peer $Peer-X$ through the "Local verification process 1"; or (b)$Peer-U$ downloaded the complete set of media chunks through "Local verification process 2" and acquire correspondent verification paths by local hashing and comparison to the root hash pair. As a result, $Peer-U$ may directly share its locally stored verification vector $D[1..u]$ to another downloading peer as the verification path for any chunk x of the latest updating period. No extra processing is required.

5.2 Tree-based scheme for VoD

The root hash of a published VoD file s, is published to the portal by the content source using the hash tree algorithm based on the complete chunk set of the file. The intermediate sharing peer, after downloading a media chunk x, computes its hash checksum h_x and stores the value into the local verification tree D as the xth leaf on the tree. The hash values of the sibling nodes along the tree path from the correspondent leaf node to the root constitute the verification path for chunk x, i.e. $s_x = tree-path(X, d, n, x)$. (See Algorithm 2). The downloading peer needs only to acquire the verification path s_x from any intermediate sharing peer, as well as the root hash s from the portal, to verify chunk x's integrity through Algorithm 3.

Root hash's formulation and publication Assume that a content source $Peer-S$ wishes to publish a VoD file X to the portal. $Peer-S$ locally produces the descriptive information $m(X)$ for the file X, and submits it, as well as the root hash $s(0)$, to the portal for publication. $s(0)$ is computed through the hash tree algorithm where the complete chunk set of the file constitute the leaves of the tree and its degree d is determined locally based on the file's scale and average storage and computation expenditure for verifying the integrity of each chunk. (See Section 6 for reference.)

$$s(0) = tree(X[1..n], H, d) \qquad (7)$$

Algorithm 2 $tree-path(T, x)$

```
$P \leftarrow \emptyset; temp \leftarrow getleaf(T, x)$
while $temp \neq T.root$ do
    $AddSibling(T, temp) to P; temp \leftarrow Parent(T, temp)$
end while
return $P$
```

Algorithm 3 $tree-verify(P, r, x, d)$

```
$p \leftarrow Lengthof(P); n \leftarrow (p-1)/(d-1); temp \leftarrow x; k \leftarrow 1$
for $i = 1$ to $n$ do
    for $j = 1$ to $d-1$ do
        $temp \leftarrow temp||P[k]; k \leftarrow k+1$
    end for
    $temp \leftarrow H(temp)$
end for
if $temp = r$ then
    return $TRUE$
else
    return $FALSE$
end if
```

Media chunks' integrity verification Assume that $Peer-D$ wishes to downloads a media chunk x of a VoD file X from an intermediate sharing peer $Peer-U$ and checks x's integrity locally.

- **Step1:** $Peer-D$ checks if the local hash tree D contains x's verification path s_x: if so, executes Step3; or Step2, otherwise.
- **Step2:** $Peer-D$ asks $Peer-U$ for verification path s_x, and verifies the validity of the latter's response s_x through the following equation, according to the root hash pair $s(0)$ from the portal:
$$tree-verify(s_x, s(0), x, d) = TRUE \tag{8}$$
If the equation holds, the path is valid, update the local hash tree D with the hash values contained by s_x and executes Step3; otherwise, give up $Peer-U$ and choose another intermediate sharing peer for chunk x.
- **Step3:** After downloading chunk x from $Peer-U$, $Peer-D$ computes its hash checksum h_x and verifies its integrity through the following equation, according to the locally stored verification path s_x:
$$h_x = s_x[1] \tag{9}$$
If the equation holds, chunk x is downloaded successfully; otherwise, give up $Peer-U$ and choose another intermediate sharing peer for downloading x.

Verification path's computation and distribution Assume that $Peer-U$ wishes to share its locally downloaded chunk x, and the correspondent verification path for other downloaders.

Intuitively, there are two possibilities: (a) $Peer - U$ downloaded the valid verification path s_x as well as x itself from another intermediate sharing peer $Peer - X$, and checked x's integrity; or (b) before downloading x, $Peer - U$ finds that a valid path is available from the current hash tree D stored locally for file X (i.e all the hash values that constitute x's verification path have been computed and stored for some other chunks downloaded earlier). As a result, $Peer - U$ may directly share its locally stored verification path s_x to another downloading peer.

6 Analysis

In Table 2 and 3, variable n stands for the number of chunks contained by an average VoD media file, which is proportional to the length of the file. m represents the average number of chunks downloaded by a participating peer, which is correlated to the coding rate of the media file and the average on-line duration of a typical user. System parameters u and d are constants determined by the system administrator, with the former depicts the number of chunks of an updating period, and the latter the degree of a VoD hash tree. In comparison, System A in the tables refers to a VoD application using a previous method with each chunk's checksum included into a vector as part of the publication data stored at portal/tracker, while system B (live streaming scenario) and C (VoD scenario) employ the proposed chain-based and tree-based schemes, respectively.

Table 2. Requirement-Satisfaction Analysis

Application	**Real-Time Verification**	**Partial Publication**	**Off-line Verification**
A (VoD-vector)	$\surd$	$\times$	$\surd$
B(Live-chain)	$\surd$	$\surd$	$-$
C(VoD-tree)	$\surd$	$\surd$	$\surd$

In system A, a downloading peer may verify the integrity of a newly downloaded chunk by using the global hash vector from the portal/tracker, therefore fulfilling the requirements of real-time verification and off-line verification. However, it is inapplicable to the live streaming scenario at all since it cannot meet the partial publication requirement as dictated in live streaming scenarios. The content source needs to compute the hash checksums for each media chunk in the form of a hash vector with length n (where n depicts the average number of chunks for a VoD file), resulting in $O(n)$ computation and communication costs for the provider in the content publication procedure, $O(n)$ storage and communication costs for the portal/tracker in checksum distribution procedure, and $O(n)$ computation and/or communication costs for the downloading peers in integrity verification procedure. On the other hand, the proposed two schemes respectively satisfy the requirements in terms of live and VoD scenarios. In both settings, the portal/tracker only needs to store the root hashes for a published program, fulfilling the partial publication requirement and reducing publication expenditures for both the provider and portal/tracker to $O(1)$, with the exception that for live streaming scenario where the root hash of a program is updated periodically (i.e. once every u chunks) by the provider the publication cost is reduced to $O(n/u)$.

In terms of the worse case computational cost for the downloading peer to perform integrity verification of a given chunk, system B and C seem to degrade the $O(1)$ performance respectively to $O(u)$ and $O(log_d(n)$ in order to verify the first chunk of a media content. However, the average verification cost for each chunk will decrease because the internal nodes' values are filled into the chain/tree as the downloaded chunk group grows, and finally approaches $O(1)$.

Table 3. Verification expenditures for each chunk

Application	**Publication**	**Verification**	**Storage**
A(VoD-vetor)	$O(n)$	$O(1)$	$O(n)$
B(Live-chain)	$O(n/u)$	$O(1)$	$O(1)$
C(VoD-tree)	$O(1)$	$O(1)$	$O(1)$

7 Conclusion

This paper proposes an efficient checksum consolidation, distribution and verification scheme to ensure chunk-level integrity protection in P2P streaming systems. The content source publishes only consolidated checksums to the network. A peer securely verifies a downloaded chunk's integrity by combining the checksums from other intermediate peers as well as the one from the content source. Specifically, a tree-based and a chain-based scheme are proposed respectively for VoD and live streaming application scenarios. Preliminary analysis shows that the proposal is highly applicable to the streaming settings and more efficient compared to previous solutions.

References

1. Vu, L., Gupta, I., Liang, J., Nahrstedt, K.: Measurement of a large-scale overlay for multimedia streaming. In: Proceedings of the 16th international symposium on High performance distributed computing, ACM (2007) 241–242
2. Y. Zhang, N. Zong, J.S., Yang, R.: Problem Statement of P2P Streaming Protocol (PPSP). (2009)
3. He, P., W.J.D.H.S.P.: Latency Hiding Algorithm for P2P File Integrity Verification. Chinese Journal of Computer Engineering **36**(15) (2010)
4. Merkle, R.: Protocols for Public Key Cryptosystems. (1980)
5. Goldreich, O.: Foundations of cryptography: Basic applications. Cambridge Univ Pr (2004)