Verifying Data Integrity in Peer-to-Peer Media Streaming[*]

Ahsan Habib†[†] Dongyan Xu[‡] Mikhail Atallah[‡] Bharat Bhargava[‡] John Chuang[†]

[†]School of Information Management and Systems
University of California, Berkeley
102 South Hall, Berkeley, CA 94720
{habib,chuang}@sims.berkeley.edu

[‡]Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{dxu,mja,bb}@cs.purdue.edu

ABSTRACT

We study data integrity verification in peer-to-peer media streaming for content distribution. Challenges include the timing constraint of streaming as well as the untrustworthiness of peers. We show the inadequacy of existing data integrity verification protocols, and propose Block-Oriented Probabilistic Verification (BOPV), an efficient protocol utilizing message digest and probabilistic verification. We then propose Tree-based Forward Digest Protocol (TFDP) to further reduce the communication overhead. A comprehensive comparison is presented by comparing the performance of existing protocols and our protocols, with respect to overhead, security assurance level, and packet loss tolerance. Finally, experimental results are presented to evaluate the performance of our protocols.

1. INTRODUCTION

A media streaming session allows the playback of media data while the data is being transmitted. Compared with traditional music file sharing, a media streaming session has longer duration and requires higher bandwidth. As a result, a media server with fixed capacity may not be able to distribute media in streaming mode to a large number of clients. Recently, peer-to-peer (P2P) real-time streaming has been proposed as a highly scalable technology to distribute content especially media data. We focus on data integrity verification during distribution of large-volume and high-quality media data using P2P real-time streaming technology. One key characteristic of P2P media streaming is that a streaming session may involve *multiple* supplying peers, due to the limited bandwidth contributed by each of them.[5]

In this media distribution scenario, a supplier peer may corrupt any block of the media data during the streaming session. As a result, media data *integrity* verification becomes a critical task, and poses the following challenges. First, unlike authentication for multicast[11, 14, 17] (one-to-many) streaming, the suppliers cannot be assumed as trusted. In P2P media streaming, packets signed by a peer may not be acceptable to other peers. Thus, a client needs a point of reference to verify the media data it receives. Second, due to the real-time constraint of media streaming, data integrity check has to be performed also in real-time. Third, the objective of checking data integrity is not only to verify that the data are not corrupted, but also to validate that the data is really what the client has requested. We note that other protection issues exist in P2P media distribution, such as how to ensure that the clients will not distribute media data to their unauthorized friends. These issues are outside the scope of this paper.

Unfortunately, existing protocols for data integrity verification are either inapplicable or too expensive for P2P media streaming. We adopt the method of *message digest*, and propose protocols that involve different *trade-off* strategies between degree of assurance and computation/communication overhead. We first propose a Block-Oriented Probabilistic Verification (BOPV) protocol for efficient data integrity verification. We show that probabilistic verification provides high assurance of data integrity and incurs significantly lower computation overhead. To further reduce communication overhead, we propose Tree-based Forward Digest Protocol (TFDP). TFDP uses Merkle signature tree, and distributes the total communication overhead over the duration of a streaming session. Both protocols work well with unreliable transport protocols. This is

[*]This work is supported in part by NSF-ITR 0085879, NSF-ANI 0219110, NSF-IIS 0209059, NSF-EIA 9903545, NSF-ISS 0219560, and CERIAS.

[†]This work was partially done when the author was at Purdue University.

achieved by using multiple hashes or Forward Error Correction (FEC) codes (applied only to digests, not data). We note that homomorphic hash function—used to verify rateless erasure codes during bulk data transfer in P2P networks[8]—can be used in our protocols. Our experiments using movies (*The Matrix, Star Wars IV*, and *From Dusk Till Dawn*) show that the proposed protocols achieve high degree of assurance in data integrity with low communication and computation overhead.

The rest of the paper is organized as follows: Section 2 surveys related work. Section 3 presents the two protocols for data integrity verification. Comparisons among different verification protocols are presented in Section 4. Section 5 provides our experimental results. Finally, Section 6 concludes this paper.

2. RELATED WORK

We discuss existing schemes that have the potential to be applied to real-time P2P streaming. We then identify the limitations of the existing schemes. To the best of our knowledge, there has been no prior study on data integrity verification in many-to-one P2P media streaming.

2.1. Digital Signature

One common way to verify data integrity is to let the server sign every packet[‡] or hash of each packet with its private key using digital signature. A user can then verify the digests using the source server's public key. The RSA signature verification has high computation overhead and not suitable for real-time applications.[15] Unlike RSA, one-time signature schemes[9, 12] incur low verification overhead and latency. These schemes are usually used to sign multicast or broadcast streams. Rohatgi proposed k-time signature scheme which is more efficient than the one-time signature scheme.[16] Still, the scheme requires 300 bytes for each signature. Moreover, these signatures are secure only for a short period of time. As a result, they are not suitable to P2P streaming, where the supplying peers may store the digests for days or weeks before the digests are supplied to other clients.

2.2. Signature Chain

Gennaro and Rohatgi introduced techniques to sign off-line and on-line digital streams.[3] The first packet of an off-line stream is signed and hash of each packet is embedded in the *next* packet. The on-line scheme signs the initial packet and embeds the public key of a one-time signature in each packet, which is used to sign the subsequent packet. Although an elegant solution, it does not tolerate packet losses and it incurs high communication overhead.

Perrig *et al.* proposed TESLA and EMSS for efficient and secure multicast.[13, 14] TESLA embeds the signature of packet p_i and the key to verify packet p_{i-1} in packet p_i. The key of packet p_i is sent in packet p_{i+1}. The adversary will see the key but it is too late to forge the signature. TESLA requires *strict ordering* of packets by the sender, which makes it inappropriate for P2P streaming where there are *multiple* senders in each session. Furthermore, if supplying peers generate keys and sign the digests like TESLA, they might not be acceptable to other clients because peers are not assumed to be trustworthy. The efficient multi-chained stream signature (EMSS) tolerates packet loss by sending multiple hashes with each packet. We also explore this option to make our protocols robust against packet losses.

2.3. Signature Tree

Wong and Lam studied data authenticity and integrity for lossy multicast streams.[17] They proposed Merkle signature tree to sign multicast streams. In their scheme, the root is signed to amortize one signature over multiple messages. Each packet contains the digests of all nodes necessary to compute the digest of the root and the signature of the root. As a result, the space requirement is rather high: 200 bytes in each packet using 1024-bit RSA for a tree of 16 packets. One of our protocols also uses Merkle tree. However, we significantly reduce the overhead by sending the digests of one subtree *before* sending any data.

Park *et al.* proposed *SAIDA* that leverages erasure codes to amortize a single signature operation over multiple packets.[11] In SAIDA, a block of a packets carries the encoded digests and signature of the block.

[‡]We use *packet* to indicate the minimum unit of media streaming, *not* the actual IP packet.

The signature and digests are recoverable, if the receiver gets any $b \leq a$ packets. This digest encoding is robust against bursty packet losses to a certain level. To reduce overhead, FEC is used to encode only digests, not data.

Both signature tree and SAIDA are designed for multicasting where the sender signs packets and the receiver trusts the sender. In our protocols, the receiver does not need to trust the senders. Unlike these signature tree and SAIDA, we do not use digital signature to reduce overheads.

2.4. Escrow Server

Horne *et al.* proposed an escrow service infrastructure to verify data in P2P file sharing environment.[6] An escrow server is responsible for file verification and for payment to peers that offer file sharing. However, it is not appropriate for streaming media dissemination, due to the unacceptable latency and overhead in verifying every single block via the escrow server.

3. PROPOSED SOLUTION

All our protocols require that a receiver collects a certain reference from a trusted authority. We name the authority as *Authentication Server* S_0 where a client authenticates itself to initiate a streaming session and obtains a point of reference that can be used in a streaming session to verify the integrity of the incoming data. If authentication is not enforced by the system, the point of reference data for media files can be distributed in the network. A peer can download the reference data from a trusted node. We define the streaming model, incentive model, and adversary model before introducing our protocols.

Streaming Model. Client P_0 requests a media file and receives the stream in real-time from a set of supplying peers $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$[§]. A media file is divided into a set of M blocks as $\mathbb{B} = \{b_1, b_2, \ldots, b_M\}$. Each block consists of l packets. We express block $b_i = \{p_{i1}, p_{i2}, \ldots, p_{il}\}$, where p_{ij} is the j-th packet of block i: A series of contiguous packets is referred to as a *block*, and a series of blocks is referred to as a *group*.

An entire block can come from one or multiple peers. Multiple peers collaborate to provide the packets of each block to the receiver, and the receiver will re-construct the block. We define a set of suppliers as active set $\mathbb{P}^{act}$ for each P2P media streaming session. The receiver assigns a sending rate to each of the active senders. The streaming session continues as far as there is no need to *switch* to a different active sending set. A switch is needed if a peer fails or the network path becomes congested.

Incentive Model. We consider peers as rational users seeking to maximize their individual utilities through their actions, as opposed to users that are strictly obedient to the protocol. In the absence of incentives, peers may choose to engage in free-riding, i.e., consuming resources without contributing any in return. This behavior has been observed in P2P file-sharing networks such as Gnutella.[2] By using appropriate incentive mechanisms, the peers can be induced to cooperate by serving as suppliers to streaming sessions.[4]

Adversary Model. In this model, any supplier of a media file may put garbage data in any segment during transmission. If a supplying peer can successfully send garbage data without getting caught by the receiving peer, the supplying peer can pretend to have a media file without actually having the file, which foils our objective that a receiver is able to verify the integrity of downloaded data. Moreover, the adversary can intentionally drop some of its own packets or others' packets to pretend that the network is congested. If the receiver does not receive enough packets to verify each segment, the suppliers will be replaced, and the adversary does not gain (incentives or satisfaction by disrupting a streaming session) anything. However, a set of adversary peers sitting at strategic locations can disrupt P2P media streaming sessions, which may eventually launch denial of service attacks on the system. This type of attack is out of scope of this paper.

Now, we present the proposed protocols to defeat the adversary during P2P media streaming.

[§]The set of suppliers is determined by a P2P lookup substrate.

3.1. Block-Oriented Probabilistic Verification (BOPV) Protocol

The Block-Oriented Probabilistic Verification (BOPV) Protocol reflects the following idea: it takes the hash of a block of packets instead of the hash of each packet, in order to reduce communication overhead. The idea of taking hash of a block has already been explored.[11, 17] We integrate probabilistic verification in this protocol that verifies selective blocks rather than all blocks during a streaming session to reduce the computation overhead. Our analysis shows that the probabilistic verification achieves high level of security assurance.

The BOPV protocol, illustrated in Figure 1, runs as follows:

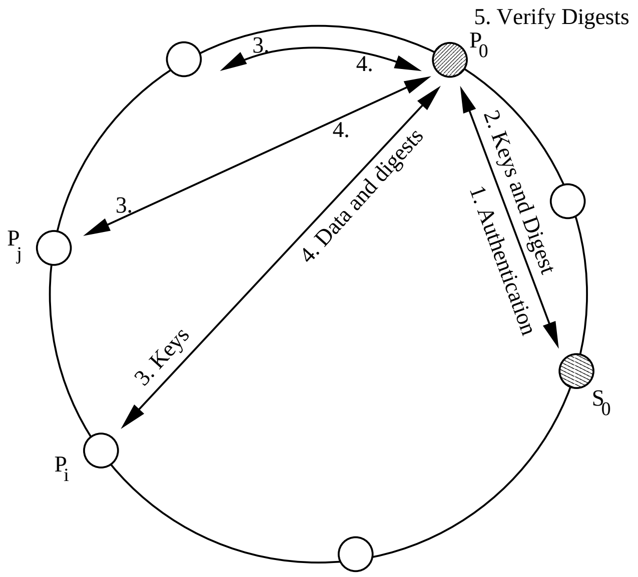


Figure 1. Steps of the BOPV protocol: P_0 is the requesting client. P_0 contacts the S_0 for authentication and communicates to different peers for the content.

- *Step 1:* Client P_0 authenticates itself with the server S_0 by sending $E_{S_0}(D_{P_0}(M_0))$, where the request message M_0 is signed by P_0 for non-repudiation. Then, it is encrypted with the public key of S_0.
- *Step 2:* The server generates a secret key $K_{i=1...M} \in \mathbb{K}$ for each block i, where $\mathbb{K}$ is set of all keys for the file. To reduce computation overhead, the server groups N blocks, and compute n message digest $\delta_{j=1...n} \in \Delta$ out of the N ($N > n$) blocks in the group, where Δ is the set of message digests of the file. The digests are computed using keyed hash.[7] These digests are used as a reference to verify the data sent later by the suppliers. Thus, the server sends $E_{P_0}(D_{S_0}(\mathbb{K}, \Delta, T))$, which is signed by the server and encrypted with P_0's public key. T is a timestamped ticket that needs to be presented to each peer for authentication. The timestamp prevents a peer from using the same ticket beyond a specific time period.
- *Step 3:* P_0 sends $E_{X_i}(K_{i=1...k} \in \mathbb{K}, T)$ to each supplier having k blocks to provide one key for each block assuming P_0 and P_i shares a common key X_i for a streaming session. An arbitrary adversary cannot fool P_0 easily because it has to obtain the keys first. Without the keys the digests are not acceptable.
- *Step 4:* Each supplier uses the keys to generate digests $\delta'_{j=1...k}$ and sends them to P_0 with the blocks $b_{j=1...k}$. These digests are required in the verification process when some of the packets are lost.
- *Step 5:* For each block, P_0 computes the digest and matches the outcome against the corresponding digest it receives from the server. If there is a match, the block is accepted, otherwise the block is rejected.

An example: If the server divides the movie *The Matrix* of size 1.3 GB into packets of size 1 KB, 26 MB of digests will be generated assuming each hash is 160 bits long. P_0 may not want to download this amount of data before starting the streaming. However, if each block contains 128 packets, the volume of digests is reduced by 128 times, which is 0.209 MB. To further reduce overhead, the server randomly selects n blocks out of the N ($N > n$) blocks to generate digests, and gives them to P_0. Each supplier peer does not know which blocks will be tested by P_0, and they send all digests to P_0. P_0 *only* verifies the blocks it gets digests from the server. Verifying 8 out of 16 blocks will finally reduce the the total digest transmission overhead to 107 KB.

Probabilistic verification. BOPV provides adjustable levels of security and reduces computation overhead. In general, if a malicious peer tampers with r blocks out of N blocks, the cheat success probability is $Pr[cheat(N,n,r)] = \frac{\binom{N-r}{n}}{\binom{N}{n}} = \frac{(N-r)! \times (N-n)!}{(N-n-r)! \times N!}$. $Pr[cheat(N,n,r)]$ defines the probability that a malicious peer can successfully send r corrupted blocks to the client without getting caught because of the probabilistic verification performed by the client.

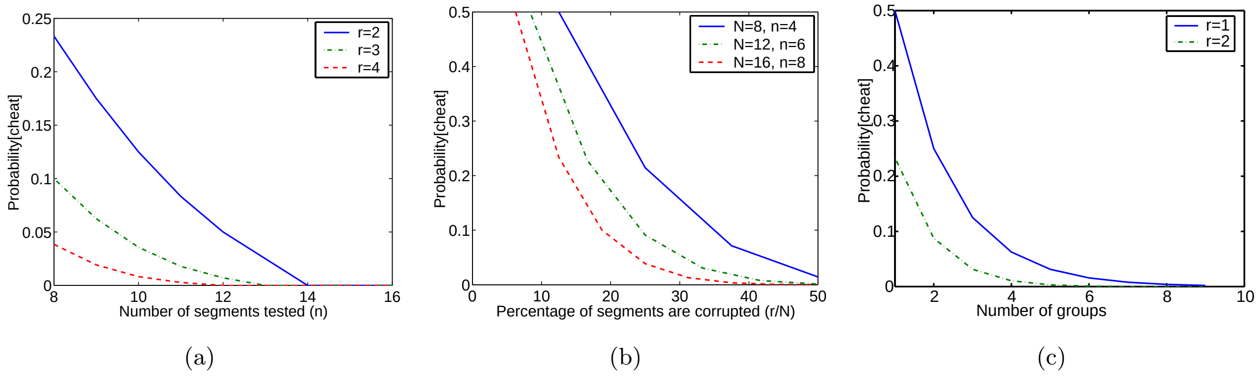


Figure 2. Success probability of cheating under different number of blocks. N is the number of blocks of a group, n is the number of blocks that are verified out of N, and r is the number of blocks an adversary corrupts in a group. (a) Increasing number of blocks to verify (n) reduces success cheating probability. (b) Success cheating probability is very low when 30% or more blocks are corrupted. (c) X-axis is the number of groups an adversary tries to cheat r blocks. The cheating probability drops exponentially in multiple groups.

Now, we show how to reduce the cheat success probability. Let $N = 16$ and $n = 8$. If a peer tampers with one block, i.e. $r = 1$, the chance to detect this is only 50%. However, if a peer tampers with 4 blocks, then more than 96% of the time P_0 will detect that Figure 2(a). This probability will reach 0.99 if $n = 9$. Again, the cheat success probability is very low when 30% or more blocks are corrupted (Figure 2(b)). Therefore, the level of data integrity assurance can be adjusted by tuning the values of n and N.

The cheat success probability drops exponentially when a redistribution peer attempts to corrupt multiple groups of data: The probability goes down to 0.002 when one block is corrupted in 10 groups. Figure 2(c) shows that the probability is 0.0008 when two blocks are corrupted in six groups. Thus, the probabilistic verification of BOPV reduces both computation and communication overhead, yet at the same time, data integrity violation can be detected with a very high probability.

One limitation of BOPV is that if any packet is lost, the requesting peer will not be able to verify the entire block containing the lost packet. To deal with packet losses, we have the following two extensions to BOPV.

Multiple Hashes (BOPV-MH). Efficient multi-chained stream signature (EMSS) achieves robustness against packet losses by sending multiple hashes (or digests) of other packets with the current packet[14] . We study how this idea performs in P2P media streaming. Here, the supplier send each packet $p_{ij} = [M_{ij}, H_{i,j+1\%l}, \ldots, H_{i,j+t\%l}]$, where t defines the loss threshold, and M_{ij} is the data of j-th packet for block

i. To verify the packets of a block b_i, peer P_0 checks which packets of the block it received and which of them are lost. When a packet is lost, its hash will be found in other packets unless total packet loss of a block exceeds the threshold t. P_0 computes hashes of packets received, and uses the hash provided by the sending peer for lost packets. If the computed digest matches the digest provided by the server S_0, the client accepts the data, otherwise it rejects the data. We can tolerate up to t lost packets out of l packets in a block yet we can still verify the integrity of the remaining $l - t$ packets.

We provide an example to illustrate the digest scheme for lossy environments. Let, let $l = 5$ (l is the number of packets in a block) and $t = 2$. P_0 receives $h(K_i, H_{i1}, H_{i2}, H_{i3}, H_{i4}, H_{i5}, K_i)$ from a trusted entity as a point of reference. A peer sends $[M_{i1}, H_{i2}, H_{i3}]$, $[M_{i2}, H_{i3}, H_{i4}]$, $[M_{i3}, H_{i4}, H_{i5}]$, $[M_{i4}, H_{i5}, H_{i1}]$, $[M_{i5}, H_{i1}, H_{i2}]$. If the first and second packets are lost, P_0 can still verify by using hashes H_{i1} and H_{i2} sent by the peer in the fourth and fifth packets, and computing hashes for rest of the packets. In this example, we can tolerate up to two lost packets out of five packets.

Forward Error Correction (BOPV-FEC). Park *et al.* used erasure code to encode digests and signatures instead of data block.[11] We apply a similar idea to BOPV. For each block, the peers encode the digests into αa ($\alpha \geq 1$) packets out of which a packets are sufficient to decode the digests. This scheme is robust against bursty packet losses because any a packets can recover the digests of all packets. However, if less than a packets are available to the client, the whole block cannot be verified. The receiver first decodes the digests, and then verifies the integrity of the received packets.

3.2. Tree-based Forward Digest Protocol (TFDP)

To further reduce the initial digest downloading overhead, we propose TFDP, or Tree-based Forward Digest Protocol. It requires to download only one digest from the server S_0. TFDP can adopt the probabilistic verification to reduce the computation overhead at the receiver.

TFDP uses Merkle tree, and is similar to *Tree-chaining* proposed by Wong and Lam[17] for multicast flows. However, our protocol does not sign the root of every subtree that belongs to each block. Instead, we only compute digests to form the Merkle tree. Another difference is that our protocol creates one tree for the entire media file, rather than a separate tree for each block.

Initially, the server generates the Merkle signature tree for a media file. The leaves of the tree are packets of a block. All non-leaf nodes of the tree represent digests of the leaves of their corresponding subtrees. The server enforces a minimum number of blocks to be stored at each supplier so that the overhead of sending extra digests is amortized over a group of blocks. During a streaming session, N_{min} digests are downloaded before downloading the original blocks. A higher N_{min} will reduce the overhead. However, it will incur longer delay in the streaming session.

Figure 3 shows a simplified example with 32 packets that are part of 8 blocks. Let P_1 be assigned to provide the digests of first two blocks, P_2 provides digests of next four, and P_3 provides the rest. From previous section, we know that a block is downloaded from a set of peers $\mathbb{P}^{act}$. P_1, P_2, and P_3 are members of $\mathbb{P}^{act}$ at different part of the streaming session. When P_0 wants to download blocks from $\mathbb{P}^{act}$, P_1 first provides all digests to compute the digest of the root. In this case, those are H_1, H_2, H_{10}, and H_{14}. P_0 computes H_9 from H_1 and H_2, H_{13} from H_9 and H_{10}, and H_{15} from H_{13} and H_{14}, and then verifies with the digest supplied by the server. If there is a match, the *belief* in H_{15} is transferred to all hashes provided by P_1. Later, the data sent by the active set $\mathbb{P}^{act}$ is verified block by block using H_1 and H_2. P_2 and P_3 act independently in a similar fashion. We now describe the steps of TFDP:

- *Step 1:* Client P_0 authenticates itself with the server S_0 by sending $E_{S_0}(D_{P_0}(M_0))$, where the request message M_0 is signed by P_0 for non-repudiation. Then, it is encrypted with the public key of S_0.
- *Step 2:* The server provides P_0 the digest of the root of the Merkle tree $E_{P_0}(D_{S_0}(D_{root}, T))$ encrypted with the public key of P_0 and signed by its private key. T is the ticket used in authentication for each session with P_i. The timestamp prevents a peer from using the same ticket beyond a specific time period.

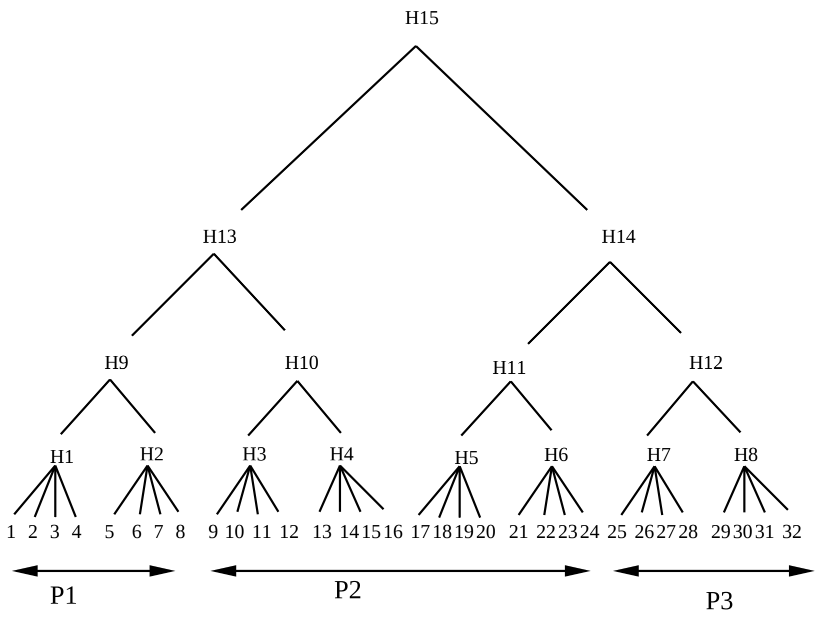


Figure 3. Tree structure of 32 packets that constitute 8 blocks. P_1, P_2, and P_3 are members of an active set at different part of streaming session. P_1 is assigned to provide digests that are required to verify first two blocks. P_1 sends H_1, H_2, H_{10}, and H_{14} to P_0. P_2 provides H_3, H_4, H_5, H_6, H_9, H_{12} and P_3 provides H_7, H_8, H_{11}, and H_{13} .

- *Step 3:* P_0 tells each supplying peer P_i of the active set $\mathbb{P}^{act}$ to forward the digests that are required to verify the N_{min} blocks assigned to the active set.
- *Step 4:* Each peer P_i provides the digests of all leaves of the subtree it has and digests of all other internal nodes to compute the root. Each peer sends $\delta'_{ij}, i = 1 \dots |\mathbb{P}^{act}|, j = 1 \dots k$ to P_0, where k is the number of digests required to verify all N_{min} blocks. These digests are obtained from the server off-line.
- *Step 5:* If the computed digest at P_0 matches the root digest obtained from the server, P_0 will allow P_i to send the data. P_0 can trust the digests of each block of the N_{min} blocks because the computed digest matches the digest of the root.
- *Step 6:* P_0 signals the peers that the digests are verified, and requests them to send media data.
- *Step 7:* The peers send media data, and P_0 can verify every block individually.

To reduce the delay in *Step 4*, we tune the value of N_{min}. For example, if N_{min} is 64 blocks, then *Step 4* downloads $N_{min} + \left\lceil \log(\frac{F}{N_{min} l}) \right\rceil$ digests, which is equal to 75 digests, i.e., 1500 Bytes for our example. Downloading this digest takes very little time for P_0.

Figure 3 is a binary tree if we exclude the leaves. Each leaf is a packet, and the parent of the leaves represent the digests of the blocks that contain the packets. The size of a block needs to be chosen carefully to ensure that it does not introduce delay to collect all packets of a block. All the blocks (internal nodes) can be arranged as a d-ary tree. The height of the tree will be $\log_d \frac{F}{l}$, where F is the size of the media file. The extra digests required to verify each block depends on the height of the tree. It requires $(d-1)\left\lceil \log_d \frac{F}{N_{min} l} \right\rceil$ digest to verify a group of N_{min} blocks. The number of extra digests required to verify a group of N_{min} blocks is minimized when $d = 2$. Higher value of N_{min} can reduce the required number of digests, however, it might increase delay to download extra digests before downloading data.

Table 1. Comparison of different data integrity verification protocols. M is total number of block in a file, l is the size of a block in packets, v is the probability that a block is verified, α is defined in Equation 1, K is the size of a key, N_{min} is the minimum number of block a peer stores, and $X = M + \frac{M}{N_{min}} \log(\frac{M}{N_{min}})$.

	Allow packet loss	Download server $\rightarrow P_0$ (Bytes)	Download $\mathbb{P} \rightarrow P_0$ (Bytes)	# of Hash computation at server	# of Hash computation at P_0	Sign at server	Verify sign at peers	Decode at P_0	Security
Tree Chaining (1024 bit)	YES	0	$20Ml\log l$ $+128Ml$	$M(2l-1)$	$M(2l-1)$	M	M	—	deterministic
BOPV	NO	$(20+K)Mv$	$20M$	Mv	Mv	—	—	—	probabilistic
BOPV+MH	YES	$(20+K)Mv$	$20Mlt$	$Mv(l+1)$	$Mv(l+1)$	—	—	—	probabilistic
BOPV + FEC	YES	$(20+K)Mv$	$20Ml\alpha$	$Mv(l+1)$	$Mv(l+1)$	—	—	M	probabilistic
TFDP	YES	20	$20Ml\alpha + 20X$	$2M-1$	$Mvl + M/N_{min}$ $\times[(N_{min}-1)$ $+\log(M/N_{min})]$	—	—	M	probabilistic
SAIDA	YES	0	$(20l+128)M\alpha$	$M(l+1)$	$M(l+1)$	M	M	M	deterministic

4. COMPARISON AND EVALUATION

It is shown that SAIDA performs better than both EMSS[14] and chaining[17] in tolerating bursty packet losses.[11] Therefore, we only compare our protocols with SAIDA[11] and Tree Chaining.[17] We evaluate the overhead of BOPV, the BOPV variations that integrate multiple hashes (BOPV-MH) and FEC codes (BOPV-FEC), as well as TFDP.

We compare the communication and computation overhead of the protocols. The communication overhead is the extra bytes per packet client P_0 needs to download from other peers and the server S_0 to verify data integrity. The computation overhead (of P_0) is due to hash computation, signature verification, and FEC decoding. Before the comparison, we show the overhead computation for each protocol.

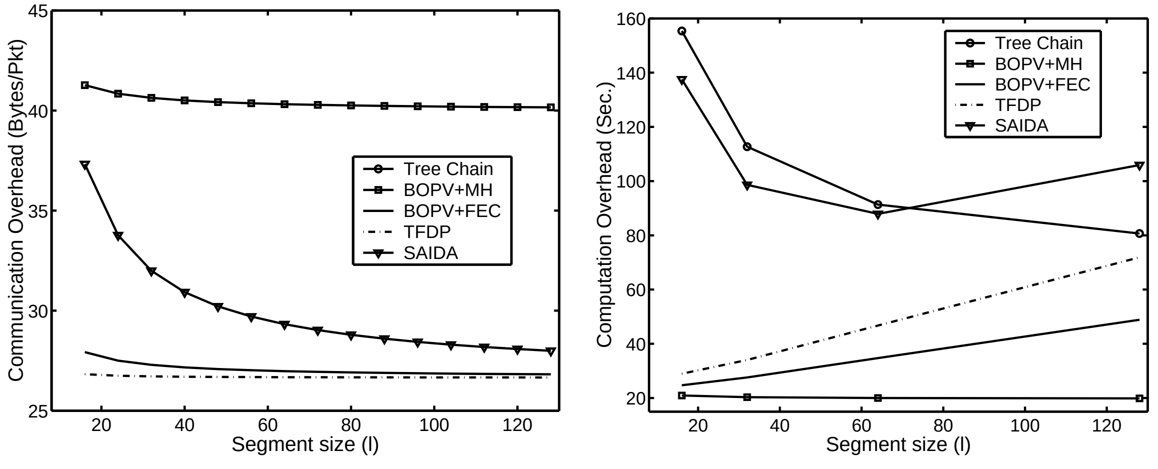
Tree Chaining needs to download the public key (usually 128 bytes) of the server to verify the signature. The receiving peer P_0 downloads $l \log l$ digests for each block, where l is the size of a block in terms of packets. Each packet carries one 1024-bit signature. Thus, for each block P_0 downloads $20l \log l + 128l$ bytes. All flavors of BOPV download one digest and one key from the server for each block. TFDP downloads only one digest (20 bytes) from the server. However, it needs $1 + \frac{1}{N_{min}} \log(\frac{M}{N_{min}})$ extra digests for each block, and digest of each block is encoded using FEC. We define α, the overhead due to FEC, as:

$$\alpha = \frac{\text{total packets sent per block}}{\text{total packets required to reconstruct the block}} \tag{1}$$

Thus, the total communication overhead of TFDP is $20(l\alpha + 1 + \frac{1}{N_{min}} \log(\frac{M}{N_{min}}))$ bytes per block. SAIDA downloads one signature per block, and it uses FEC. Thus, it incurs $(20l + 128)\alpha$ bytes of overhead for each block. Tree Chaining and TFDP require similar amount of digest computation because both of them use Merkle tree.

BOPV-MH requires only $M(l + 1)$ hash computation. The Tree Chaining has M subtrees, and each tree incurs $2l-1$ hash functions, which is close to the load of TFDP. TFDP needs to compute extra digests for each N_{min} blocks verification. The number of extra digests computation is $M/N_{min}[(N_{min} - 1) + \log(M/N_{min})]$, and it verifies each packet with probability v. Thus, total computation overhead at the receiver is $Mlv + M/N_{min}[(N_{min} - 1) + \log(M/N_{min})]$. Table 1 summarizes the comparison results. Now, we compare the communication and computation overhead of these protocols by using them to distribute the movie *The Matrix* in a P2P network.

Communication Overhead. Figure 4 shows the performance results of different protocols based on the movie *The Matrix*. Figure 4(a) shows that the communication overhead can be reduced significantly, if FEC is used to encode digests and signatures. The Tree Chaining has extremely high communication overhead (208 bytes, for l=16, not shown in Figure 4). TFDP and BOPV-FEC incur less overhead than SAIDA. TFDP reduces the overhead by combining N_{min} blocks together to make a group. Then, it downloads the necessary digests to verify all blocks of the group. This reduces the height of the verification tree from $\log M$ to $\log \frac{M}{N_{min}}$. The difference in communication overhead narrows when the block size gets larger.



(a) Communication (b) Computation

Figure 4. Overheads among Tree Chaining, BOPV with multiple hashes, BOPV with FEC, TFDP, and SAIDA for movie *The Matrix* of size 1.3 GB. The communication overhead is shown per packet and the computation overhead is for the entire file.

Computation Overhead. We use openSSL crypto library to calculate SHA-1 hash, RSA sign, and RSA verification. Cauchy-based Reed-Solomon code is used to encode digests in our protocols and in SAIDA. Figure 4(b) shows that the BOPV with multiple hashes has the lowest computation overhead. If FEC is used, the computation overhead increases with the block size, because the decoder needs to decode more packets within a block. The computation overhead in Tree Chaining is reduced by caching digests carried by previous packets. This cache is used to verify upcoming packets of a block. However, its high communication overhead makes this solution hard to deploy in P2P real-time streaming. SAIDA has higher computation overhead than TFDP because SAIDA has to verify the signature for each block, which takes longer time than verifying a digest. TFDP has higher computation overhead than BOPV; however, we prefer TFDP over BOPV because, unlike the latter, TFDP reduces the initial communication overhead between the client and the authentication server.

5. EXPERIMENTAL RESULTS

We perform P2P streaming experiments using both ns-2 simulations[10] and real-world implementation. In our simulation, one client requests and receives streaming media from five peers. Like in SAIDA, we use a *Two-state Markov* loss model to introduce bursty packet loss. The parameters of the Markov model are $Pr\{\text{no loss}\} = 0.95$ and $Pr\{\text{loss}\} = 0.05$. The shared link incurs a packet loss rate of 25%. We calculate the fraction of verifiable packets by

$$V = \frac{1}{M}\sum_{i=1}^{M}\frac{\text{number of verifiable packets in block}\, i}{\text{number of packets received in block}\, i} \tag{2}$$

We compare SAIDA and TFDP in the simulation. The simulation results are shown in Figure 5. The digests and signatures are encoded to tolerate 37.5% packet loss rate. We observe that due to burstiness, some blocks have low fraction of verifiable packets. The reason why TFDP performs better is that SAIDA sends slightly more data than TFDP due to RSA signature for each block.

We have developed a P2P media streaming system called *PROMISE*.[5] Our system monitors network dynamics, quality of connections from suppliers to receivers, as well as availability of peers, in order to maintain full media playback quality on the client side. Particularly, the set of active peers in each streaming

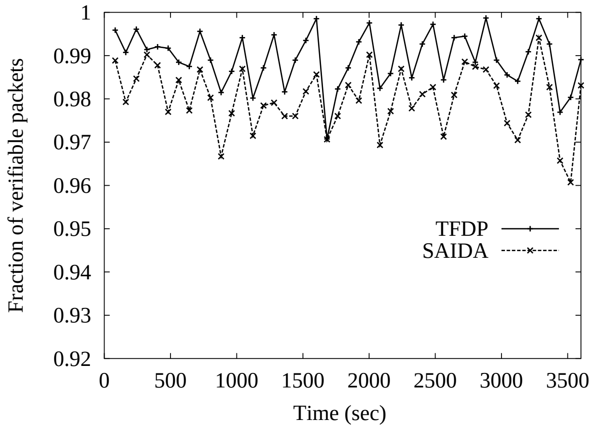


Figure 5. Fraction of verifiable packets at the receiver during simulation. More than 97% received packets are verified using TFDP.

session may change dynamically, so that the fluctuation of network and peer conditions will not affect the client-side aggregated media streaming rate. Each of the proposed data integrity verification protocols can be plugged into PROMISE. We evaluate our system by conducting experiments in the wide-area *PlanetLab*[1] testbed.

Figure 6 shows two sets of results from two PlanetLab-based experiments. Both can tolerate up to 20% packet loss by using FEC. If the loss rate is more than 20%, an entire block of packets will not be verifiable. TFDP is used in both experiments. In Experiment 1 (exp1), TFDP can verify almost all the packets throughout the experiment and thus all packets are verified most of the time. Sometimes, the loss goes as high as 40% and the probability goes down to 0.9. In this case, we have to discard the blocks. Experiment 2 (exp2) experiences a few more glitches than Experiment 1. If the loss continues for a while, the peer(s) on the congested path will be replaced by other peers, thanks to the adaptivity of our *PROMISE* system. The wide-area experimental results show that with FEC, fraction of verifiable packets is very high when TFDP is used.

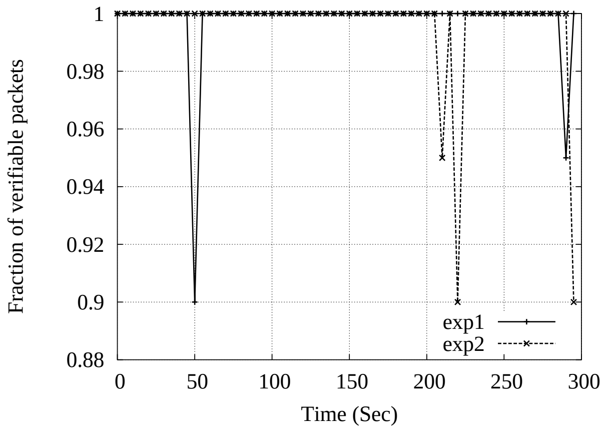


Figure 6. Fraction of verifiable packets in two P2P media streaming experiments in the wide-area PlanetLab test-bed. Due to bursty loss, the fraction of verifiable packets goes down by as high as 10%.

We use video traces of two movies (*Star Wars IV* and *From Dusk Till Dawn*) encoded using MPEG-4 to study how FEC overhead α impacts on the performance of our protocols. The traces have the information of frame number, frame type (I, P, or B), frame play-out time, and frame length in bytes. We stream the first

20 minutes of each movie, and both movies have a frame rate of 25 frames per second. For each streaming session, we record the arrival time of each single packet. Then, we determine the number of frames that would have missed their deadlines. In this experiment, we vary the overhead due to FEC to tolerate packet loss. We calculate the number of segments that cannot be decoded because more than (α - 1)% of the packets are lost or corrupted. For each undecodable segment, we consider all of its packets as lost and count the number of frames. Figure 7 shows the number of undecoded blocks for two different experiments. It shows that with all blocks of *Star Wars IV* can be decoded when $\alpha = 1.2$ in both cases. However, without the FEC, 47 blocks are wasted for exp 1 and 67 blocks are wasted in exp 2. Thus, with 20% FEC overhead, the number of undecoded blocks can be reduced significantly. Experiments with the movie *From Dusk Till Dawn* produce similar results.

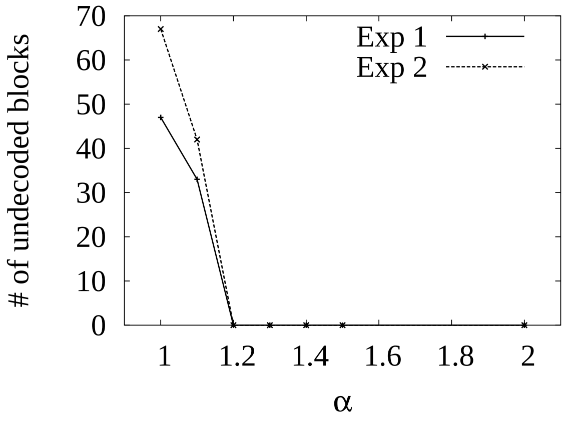


Figure 7. Number of undecoded blocks comparing to the level of redundant data due to FEC. The result is shown for the movie *Star Wars IV* in wide area Planet-Lab test-bed.

6. CONCLUSION

In P2P media distribution, data integrity verification is a critical requirement. Existing data integrity verification protocols are either inapplicable or too expensive for P2P media streaming. We propose simple and efficient protocols to verify data integrity in real-time during streaming sessions that involve multiple supplying peers. Our probabilistic packet verification protocol (BOPV) provides good trade-off between integrity assurance and verification overhead. Our Tree-based Forward Digest Protocol (TFDP) incurs even lower communication overhead and tolerates packet losses with moderate computation overhead. Our real-world and simulation experiments demonstrate the effectiveness of the proposed protocols. Particularly, TFDP is able to verify close to 100% of the media data in a P2P streaming session, even under a packet loss rate of 20%.

REFERENCES

1. Planetlab testbed. http://www.planet-lab.org/, 2004.
2. E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), Oct. 2000.
3. R. Gennaro and P. Rohatgi. How to sign digital streams. Technical report, IBM T. J. Watson research center, 1997.
4. A. Habib and J. Chuang. Incentive mechanism for Peer-to-Peer media streaming. In *proceedings International Workshop on Quality of Service (IWQoS '04)*, pages 171–180, Montreal, Canada, June, 2004.
5. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using CollectCast. In *proceedings ACM Multimedia (MM '03)*, Nov. 2003.

6. B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *proceedings ACM Electronic Commerce (EC '01)*, Oct. 2001.
7. H. Krawcayk, M. Bellare, and R. Canetti. HMAC: keyed-hashing for message authentication, RFC 2104, 1997.
8. M. N. Krohn, M. J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *proceedings IEEE Symposium on Security and Privacy*, Oakland, California, May, 2004.
9. L. Lamport. Constructing digital signatures from a one-way function. Technical report, SRI-CSL-98, SRI International Computer Science Laboratory, Oct. 1979.
10. S. McCanne and S. Floyd. Network simulator ns-2. http://www.isi.edu/nsnam/ns/, 1997.
11. J. M. Park, E. Chong, and H. Siegel. Efficient multicast packet authentication using signature amortization. In *proceedings IEEE Symposium on Security and Privacy (S&P)*, May 2002.
12. A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *proceedings ACM Conference on Computer and Communications Security (CCS '01)*, pages 28–37, Philadelphia, PA, Nov. 2001.
13. A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *proceedings Network and Distributed System Security Symposium, (NDSS '01)*, San Diego, CA, Feb. 2001.
14. A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *proceedings IEEE Symposium on Security and Privacy (S&P '00)*, pages 56–73, Nov. 2000.
15. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public key cryptosystems. *Commnunication of the ACM*, pages 120–126, Feb. 1978.
16. P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In *proceedings ACM Conference on Computer and Communications Security (CCS '01)*, pages 93–100, Nov. 1999.
17. C. Wong and S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, Aug. 1999.