Verifying Data Integrity in Peer-to-Peer Video Streaming

Ahsan Habib, Dongyan Xu, Mikhail Atallah, Bharat Bhargava
CERIAS and Department of Computer Sciences Purdue University, West Lafayette, IN 47907
{habib, dxu, mja, bb}@cs.purdue.edu

Abstract

In this paper, we study the verification of data integrity during peer-to-peer video streaming sessions. Challenges include the timing constraint of streaming, as well as the untrustworthiness of peers. We show the inadequacy of existing authentication protocols and propose a more efficient protocol which utilizes message digest and probabilistic verification. We then propose One Time Digest Protocol (OTDP) and Tree-based Forward Digest Protocol (TFDP) to further reduce the communication overhead. Finally, a comprehensive comparison is presented comparing the performance of existing protocols and our protocols, with respect to overhead, security assurance level, and packet loss tolerance.

1 Introduction

Consider the following media distribution system: a central server (say, Hollywood) first starts the streaming distribution of some media data. When there are sufficient clients (or 'peers') in the system that have received the media data, they will begin distributing the media to other peers. However, the distribution is supervised by Hollywood: it authenticates requesting peers and gives them credentials to obtain media streaming from other peers. Meanwhile, the supplying peers will perform media streaming only if proper credentials are presented. Due to limited bandwidth of peers, a peer-to-peer (P2P) streaming session may involve more than one supplying peer.

In such a system, data integrity verification poses challenges. First, the peers cannot be assumed trust-worthy. Thus, the requesting peer needs a point of reference to verify the data it receives from other peers. Second, the objective of checking data integrity is not only to verify that the data are not corrupted, but also to validate that the data are really what one has requested. For example, if a peer requests the movie *Matrix*, data integrity verification should ensure that it is getting *uncorrupted* data of *Matrix*, not those of *Star Wars*. Third, due to the timing constraint of streaming, the integrity check has to be performed efficiently without causing significant delay.

Unfortunately, existing protocols for data integrity verification are either expensive or inapplicable for P2P streaming. A comprehensive analysis and comparison will be presented in Section 4. In this paper, we adopt

the method of message digest, and propose three protocols that involve different trade-off strategies between degree of assurance, computation and communication overhead. We show that probabilistic verification provides high assurance of data integrity and incurs low computation overhead. We propose One Time Digest Protocol (OTDP) and Tree-based Forward Digest Protocol (TFDP) which further reduce the communication overhead. Our protocols work well with unreliable transport protocols. This is achieved by using multiple hashes or Forward Error Correction (FEC) codes (applied only to digests, not data). By both analysis and simulation (using the 1.3 GB *Matrix* movie), we show that our protocols outperform existing protocols.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents our protocols and discusses protocol parameter configuration in order to obtain desired level of security. Analysis and simulations are presented in Section 4. Finally, Section 5 concludes this paper.

2 Related Work

One common way to verify data integrity is to let the server sign every packet or the hash of every packet with its private key using RSA [9] digital signature. A peer then caches the packets as well as the signatures. The signatures will be provided to other peers that request the data, and verified via server's public key. RSA signature verification incurs high computation overhead at the peers. Although one-time and fast signature schemes such as [6, 8] can reduce computation and/or communication overhead, these signatures are only secure for a short period of time. Rohatgi [10] proposed k-time signature scheme which is more efficient than one-time signature scheme. Still, the scheme requires 300 bytes for each signature.

Wong and Lam [11] studied data authenticity and integrity for lossy multicast flows. They proposed Merkele's signature tree to sign multicast stream. In this 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 high: 200 bytes in each packet using 1024-bit RSA for a tree of 16 packets. Our TFDP also uses Merkele's tree. However, we significantly reduce the overhead by first sending the digests of one subtree before sending any data.

Perrig *et al.* [7] proposed TESLA and EMSS for efficient and secure multicast. 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, which makes it inappropriate for P2P streaming where packets are transmitted from *multiple* supplying peers. If supplying peers generate keys and sign the digests like TESLA, it might not be acceptable to other receiving peers. The efficient multi-chained stream signature (EMSS) tolerates packet loss by sending multiple hashes with each packet.

Park *et al.* [5] proposed *SAIDA* which leverages erasure codes to amortize a single signature operation over multiple packets. In SAIDA, a block of a packets carries the encoded digests and signature of the block. The signature and digests are recoverable, if the receiver gets any b packets. The main advantage of FEC is that it is robust against bursty packet losses. In our protocols, FEC codes are used to encode digests, not data. We show that our protocols incur much lower overhead than SAIDA yet it achieves satisfactory loss resilience.

Horne *et al.* [4] proposed an escrow service infrastructure to verify data in P2P file sharing environment. An escrow server is responsible for file verification and for payment to peers that offer file sharing. However, it is not appropriate for video streaming due to the unacceptable latency and overhead of verifying every single segment via the escrow server. Castro *et al.* [2] address different security issues in P2P network routing. They show that *self-certifying* data can help secure P2P routing. However, they do not consider P2P transmission of time-sensitive data. To the best of our knowledge, there has been no prior study specifically targeting data integrity verification for P2P video streaming.

3 Proposed Solution

We use message digest instead of digital signature, because the latter has high computation overhead and generates long signatures. All our protocols require that a requesting peer collects certain references from the central server for data integrity verification. And they all are based on the following model.

Peer P_0 requests a media file and receives the stream from a set of peers $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$, where m is the total number of peers that participate in the streaming session. We assume a media file is divided into a set of M segments[1] as $\mathbb{S} = \{s_1, s_2, \ldots, s_M\}$. Each segment consists of l packets. We express segment $s_i = \{p_{i1}, p_{i2}, \ldots, p_{il}\}$, where p_{ij} is the j-th packet of segment i. The hash of a segment is calculated with a secret key such as $H(s_i, K) = h(K, s_i, K)$, where h denotes a hash function.

3.1 Block-oriented Probabilistic Verification (BOPV) Protocol

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

Step 1: Peer P_0 authenticates itself to the central server Hollywood.

Step 2: The server provides P_0 a secret key $K_i \in \mathbb{K}$ for each segment i and message digest of the segment as

$$D_i = h(K_i, H_{i1}, H_{i2}, \ldots, H_{il}, K_i), \tag{1}$$

where H_{ij} is the hash of packet j of segment i. The server groups the segments and provides P_0 only n digests

[1] we use block and segment interchangeably

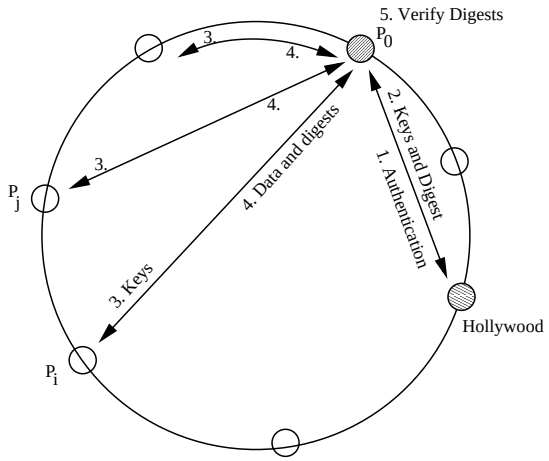


Figure 1: Steps of the protocol to receive data and verify digests by peer P_0.

out of the N segments in each group. These digests are used as a reference to verify the data downloaded from other peers. The authentication is done securely.

Step 3: P_0 gives each supplying peer one or more keys depending on how many segments the peers will provide.

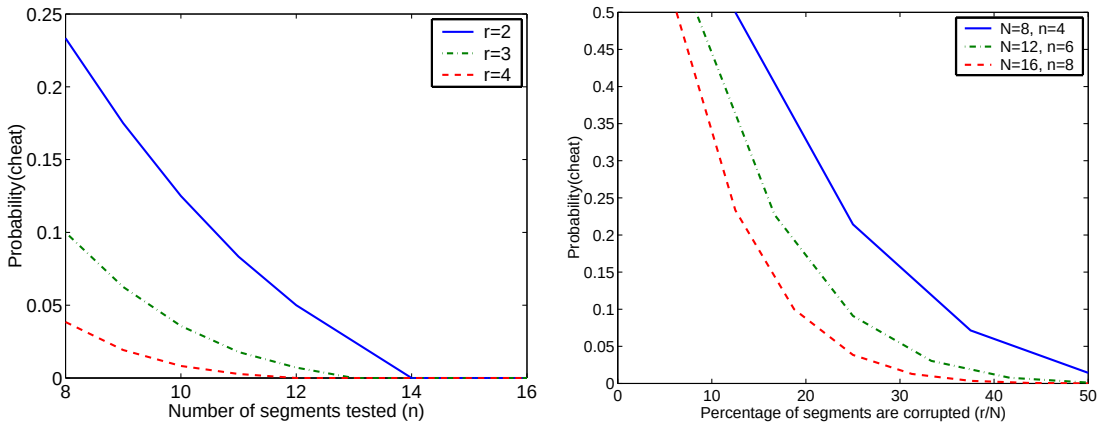
Step 4: Each supplying peer uses the keys to generate digests and sends them to P_0 with the segments.

Step 5: For verification, P_0 computes the hash function (Equation (1)) itself and matches the results against the digests it receives from the server. If there is a match, all packets in the segment are verified.

P_0 can generate those keys by itself instead of getting it from the server. However, it requires P_0 to compute two hashes to verify each segment.

An example: If the server divides the movie *Matrix* of size 1.3 GB into segments of size 1 KB (to tolerate loss), this generates 20 MB digest 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 segment contains 16 packets, Equation (1) reduces the volume of digests by 16 times. To further reduce the overhead, the server forms segment groups, with each group containing N segments. For each group, the server randomly selects n segments out of the N segments to generate digests and gives them to P_0. Each supplying peer does not know which segments will be tested by P_0 and they send digests of all segments. P_0 verifies only the segments it gets digests from the server. Verifying 8 out of 16 segments will reduce the communication cost by 200%. Thus, the 20 MB communication overhead is reduced to 0.625 MB. If we further increase the segment size to 128, we can lower the integrity verification traffic to 80 KB.

The probabilistic verification provides adjustable level of security and reduces computation overhead. In general, if a peer wants to tamper with r segments out of $(N - n)$ segments, the probability of successful cheating is $Pr(cheat) = \frac{\binom{N-r}{n}}{\binom{N}{n}} = \frac{(N-r)!\times(N-n)!}{(N-n-r)!\times N!}$. Figure 2 shows how the probability to cheat varies with number of segments verified at peer P_0 and percentage of corrupted segments a malicious peer may try to send. Let N=16 and



(a) (b)

Figure 2: Probability of cheating without getting caught in different number of segments. (a) Increasing number of segment to test (b) Increasing percentage of corrupted segments.

n=8. If a peer tampers with one segment, the chance of being detected is only 50%. However, if a peer tampers with 4 segments, then more than 96% chance P_0 will detect that. In other words, P_0 will get 12 good segments (out of every 16 segments) with probability 0.96. This probability will reach 0.99 if n=9. Therefore, the level of security can be adjusted by tuning the values of n and N.

One limitation of the above protocol is: if any packet is lost, the requesting peer will not be able to verify the entire segment containing the lost packet. An adversary peer can intentionally drop one packet from each segment and the whole streaming process is vulnerable. To deal with packet losses due to an unreliable transport protocol, we apply the following two techniques.

Multiple Hashes (MH): EMSS [7] achieves robustness against packet losses by sending multiple hashes of other packets with the current packet. We explore the similar idea to achieve robustness of our scheme. In this approach, the peers 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 segment i. Like Golle *et al.* [3], we can insert hashes in strategic locations in a segment so that the chain of packets are more resilient to bursty packet losses.

To verify the packets of a segment s_i, peer P_0 checks which packets of the segment 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 segment exceeds the threshold t. P_0 computes hashes of packets received and uses the hash provided by the sending peer for lost packets. These values are plugged in Equation (1) and if this digest matches the digest provided by the server, the peer accepts the data otherwise it rejects them.

Forward Error Correction (FEC): Park *et al.* [5] introduced erasure code to encode hashes and signatures in-

stead of data block. We apply similar idea to encode the digests using Reed-Solomon code. Efficient implementation of Reed-Solomon code has been reported in [1]. For each segment, the peers encode the digests into a packets out of which b packets are sufficient to decode the digests. This scheme is robust against bursty packet losses because any b packets can recover the digests of all a packets. With these digests, P_0 verifies the integrity of the received packets of a segment using Equation (1).

We apply FEC to all our protocols. In Section 4, we compare the performance of our protocols (with FEC) and SAIDA.

3.2 One Time Digest Protocol (OTDP)

To eliminate the downloading of digests in *Step 1* of Figure 1, we propose the One Time Digest Protocol (OTDP). In this protocol, the server generates the digests with keyed hash as shown in Equation (1) for a set of keys $\mathbb{K}$. The server distributes all digests to different peers off-line. The keys are not given to the peers. A peer can not alter any digest because it does not know the keys. The OTDP modifies the basic protocol in Figure 1 as follows:

Step 1: When P_0 requests a media file, it searches the P2P network to determine the supplying peers. P_0 authenticates itself with the server.

Step 2: The server provides P_0 a set of keys based on the search results.

Step 3: P_0 tells peers to send data and digests.

Step 4: Each peer P_i sends both data and digests to P_0.

Step 5: P_0 verifies each segment with appropriate keys. P_0 maintains some backup peers if some peers fail to provide data for any reason.

In OTDP, P_0 downloads only a set of keys which is fairly small n volume comparing with the digests of all segments. The integrity verification is secure due to the property of secure hash function. Error correction codes are used to protect digests. The probabilistic verification can be used to reduce computation overhead of P_0. The limitation of OTDP is that one digest can be used only once. When a set of keys is revealed to a peer, these keys can not be used later; otherwise a peer can forge digest. However, the cache of movie data that a peer has is reusable. When a peer wants to be a provider, it collects a fresh set of digests from the server off-line. The server is required to have an efficient key management scheme to assign keys to different parts of a movie. When some segments are used in streaming, the server will have to invalidate those keys and digests.

3.3 Tree-based Forward Digest Protocol (TFDP)

To avoid the download as in *Step 1* of the BOPV protocol and to reuse the same hashes, we propose Tree-based Forward Digest protocol. This protocol uses Merkele's tree and is similar to *Tree-chaining* proposed by Wong and

Lam [11] for multicast flows. However, our protocol does not sign the root of every subtree belongs to each segment. We only compute digests to form the Merkele's tree. Another difference is that our protocol creates one tree for a media file, instead of a separate tree for each segment. In Section 4, we show that this tree significantly reduces communication overhead.

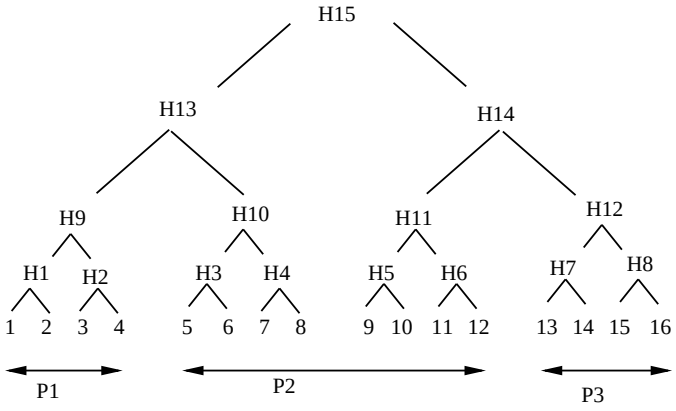


Figure 3: Tree structure of 16 segments where P_1 has first four, P_2 has next 8, P_3 has the rest. Before streaming, P_1 sends the following digests: H_1, H_2, H_3, H_4, H_{10}, and H_{14}.

Initially, the server generates the Merkele's signature tree for a media file. The leaves of the tree can be a segment (a set of packets) or a single packet. All non-leaf nodes of the tree represent digests of leaves of their corresponding subtrees. The server imposes and enforces a minimum number of segments N_{min} a peer needs to cache. And the number of segments cached by a peer is always a multiple of N_{min}. We provide a simplified example with 16 segments and $N_{min} = 4$. P_1 has first four, P_2 has next 8, P_3 has the rest as shown in Figure 3. When P_0 wants to down segments from P_1, P_1 first provides all digests of the segments it has H_1, H_2, H_3, H_4 and other digests to compute the digest of the root. In this case, those are H_{10}, and H_{14}. P_0 computes H_{15} with these information and then verifies with the digest supplied by the server. If there is a match, the *belief* in H_{15} is transported to all hashes provided by P_1. The data sent by P_1 can be verified segment by segment using these digests. The TFDP runs as follows:

Step 1: P_0 authenticates itself to the server.

Step 2: The server provides the peer credentials and the digest of the root of the Merkele's tree.

Step 3: P_0 tells a peer P_i to forward the digests that are used to verify the supplied data.

Step 4: Each peer P_i provides the digests of all leaves of the subtree it has and digests of all other nodes to compute the root. These are obtained from the server.

Step 5: If the computed digest at P_0 matches the digest obtained from the server, P_0 will allow P_i to send the data. P_0 can trust the digests of each segment sent by P_i because the computed digest matches the digest of the root.

Step 6: P_i sends data and P_0 can verify every segment individually.

To reduce the delay in *Step 4*, we can tune the value of N_{min}. For example, if N_{min} is 64 segments and each segment contains 16 packets, then *Step 4* downloads $64 \times 16 + \log(\frac{M}{N_{min}})$ digests, which is equal to 1035 digests with a volume of 20 KB for our example. Downloading this digest takes very little time for P_0. All digests are downloaded using TCP to ensure none of them is lost. The communication overhead is proportional to the height of the tree.

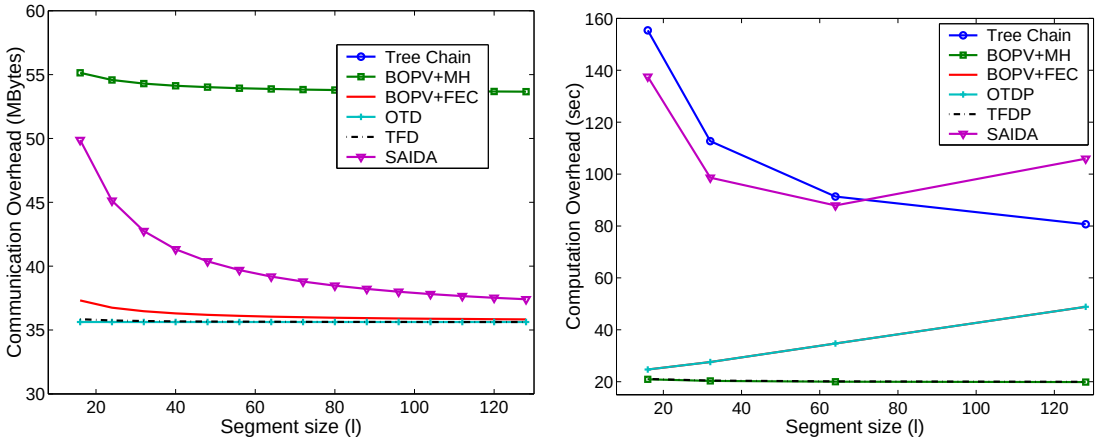
The main advantage of TFDP is that downloading digests is distributed over all peers and the server only provides the digest of the root. This scheme does not use separate key for each segment or peer. Instead, a unique identifier is used for each movie to compute the digests.

	Allow packet loss	Download server $\rightarrow P_0$ (Bytes)	Download $\mathbb{P} \rightarrow P_0$ (Bytes)	# of Hash at server	# of Hash at P_0	Sign at server	Verify sign at peers	Decode at at P_0	Security
Tree chaining	YES	0	$20Ml \log l + 128Ml$	$M(2l-1)$	$M(2l-1)$	M	M	—	100%
BOPV	NO	$(20+K)Mv$	$20M$	Mv	Mv	—	—	—	variable
BOPV+MH	YES	$(20+K)Mv$	$20Ml(t-1)$	$Mv(l+1)$	$Mv(l+1)$	—	—	—	variable
BOPV + FEC	YES	$(20+K)Mv$	$20Ml\alpha$	$Mv(l+1)$	$Mv(l+1)$	—	—	M/α	variable
OTDP	YES	KP_m	$20Ml\alpha$	$M(l+1)$	$Mv(l+1)$	—	—	M/α	variable
TFDP	YES	20	$20X$	$2Ml$	Xv	—	—	—	variable
SAIDA	YES	0	$(20l+128)M\alpha$	$M(l+1)$	$M(l+1)$	M	M	M/α	100%

Table 1: Comparison among different schemes to authenticate a stream. M is total number of segment in a file, l is the size of a segment in packets. $v = \frac{n}{N}$, probability to verify a segment. $\alpha = \frac{\text{total packets sent per block}}{\text{total packets need to reconstruct the block}}$, K is the size of a key, N_{min} is the minimum number of segment a peer caches, and $X = Ml + \frac{M}{N_{min}} \log(\frac{M}{N_{min}l})$.

4 Analysis and Results

In [5], the authors show that SAIDA performs better than EMSS [7] and augmented chaining [3] in bursty packet loss tolerance. In this paper, we compare our protocols with SAIDA and Tree chaining [11]. We evaluate the overhead of Block-Oriented Probabilistic Verification (BOPV) with its variations that integrate multiple hashes (MH) and FEC codes. The One Time Digest Protocol (OTDP) and Tree-based Forward Digest Protocol (TFDP) use FEC to achieve robustness against bursty packet losses. Table 1 presents an analytic comparison among these schemes. Before discussing the comparison, we first describe the setup. We use openSSL crypto library to calculate SHA-1 hash, RSA sign, and RSA verify. We use the Cauchy-based Reed-Solomon code [1] as forward error correction code to evaluate our protocols and SAIDA. The computation time for hash, sign, verify, encode, and decode is obtained using a 700 MHz PC with 256 MB RAM running Linux without any background process.



(a) Communication overhead

(b) Computation overhead

Figure 4: Overheads among Tree chaining, BOPV with multiple hashes and FEC, OTDP, TFDP, and SAIDA for movie *Matrix* of size, F=1.3 GB. The BOPV+FEC and OTDP have same computation overhead. Left figure does not show Tree chaining to highlight others.

Figure 4 shows the analytical comparison using Table 1 for the movie *Matrix*. The communication overhead is the total volume of digests P_0 needs to download from other peers and the server. The computation overhead is for hash computation, signature verification, and FEC decoding. The figure shows that communication overhead can be reduced significantly if FEC is used to encode digests and signatures. The FEC increases the computation overhead when segment size grows, because it needs to decode more packets within a block. The Tree chaining has extremely high communication overhead (260 MB, for l=16, not shown in Figure 4). Its computation overhead is reduced by caching digests carried by previous packets and using them to verify upcoming packets of a block. The BOPV with multiple hashes has the lowest computation overhead. Notice that, it has significant communication overhead comparing to others with FEC. The SAIDA has low communication overhead, however, the cost of verifying signature increases its computation overhead. The TFDP has slightly high computation overhead than BOPV+MH because TFDP sends few more digests for every N_{min} segments so that the receiver the verify the digest of the root.

We also conduct simulations using the *ns-2* simulator. In this simulation, one peer receives streaming media from five peers at the same time. The inbound link of the requesting peer is lossy. As SAIDA, we use *Two-state Markov* loss model (modified version of *ns-2*) to introduce bursty packet loss in the shared link. The parameters of Markov model is $Pr\{\text{no loss}\} = 0.95$ and $Pr\{\text{loss}\} = 0.05$. The shared link incurs packet loss rate of 25%. We calculate the fraction of verifiable packets by $\frac{1}{M}\sum_{i=1}^{M} \frac{\text{number of verifiable packets in segment}_i}{\text{number of packets received in segment}_i}$, for both SAIDA and OTDP. We compare SAIDA and OTDP in the simulation. The TFDP uses TCP to download digests and thus it can verify all packets received. The outcome of the simulation is shown in Figure 5. The digests and signatures are encoded to tolerate 37.5% packet loss rate. We observe that due to burstiness, some segments have low packet

verifi ability. The reason why OTDP performs better is that SAIDA sends slightly more data than OTDP due to RSA signature for each segment.

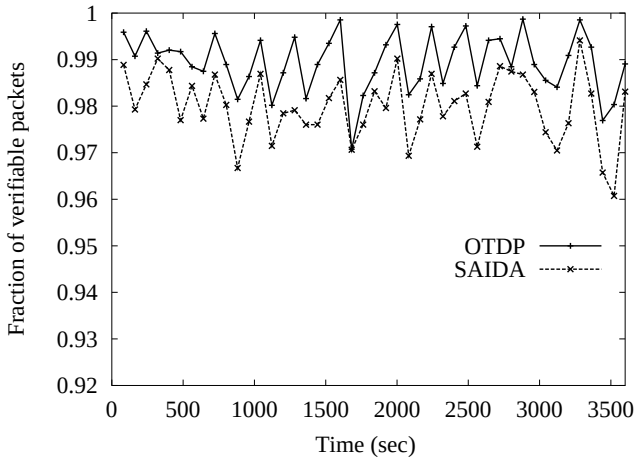


Figure 5: Packets verifi cation probability at R_0.

5 Conclusion

For P2P video streaming, data integrity verifi cation is an important security issue. However, it receives less attention than other P2P security issues. In this paper, we propose effi cient protocols to verify data integrity during P2P video streaming sessions. Our probabilistic packet verifi cation protocol tunes the security and corresponding overhead. The proposed One Time Digest Protocol (OTDP) and Tree-based Forward Digest Protocol (TFDP) have very low communication overhead and tolerate high packet losses with reasonable computation overhead. The FEC codes can reduce the communication overhead. However, we show that it increases the computation overhead when many packets are aggregated into one block in order to amortize a signature over large block. Our simulation shows that a peer can verify 97% of packets even under a packet loss rate of 25%.

References

[1] Cauchy-based reed-solomon codes. available at http://www.icsi.berkeley.edu/~luby/.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security routing for structured peer-to-peer overlay networks. In *proc. Symposium on Operating Systems Design and Implementation (OSDI'02)*, Dec 2002.

[3] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *proc. Network and Distributed System Security Symposium (NDSS '01)*, pages 13–22, Feb. 2001.

[4] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *proc. ACM Electronic Commerce (EC)*, Oct 2001.

[5] J. M. Park, E. Chong, and H. Siegel. Effi cient multicast packet authentication using signature amortization. In *proc. IEEE Symposium on Security and Privacy*, May 2002.

[6] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *proc. ACM Conference on Computer and Communications Security*, pages 28–37, Philadelphia, PA, Nov 2001.

[7] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Effi cient authentication and signing of multicast streams over lossy channels. In *proc. IEEE Symposium on Security and Privacy*, pages 56–73, Nov 2000.

[8] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *proc. 7th Australasian Conference ACSIP*, Sept 2002.

[9] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public key cryptosystems. In *Commnunication of the ACM*, pages 120–126, Feb 1978.

[10] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In *proc. ACM Conference on Computer and Communications Security*, pages 93–100, Nov 1999.

[11] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *proc. International Conference on Network Protocol (ICNP)*, Oct 1998.