Verifying Data Integrity in Peer-to-Peer Media Streaming

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

Abstract

We study the verification of data integrity during peer-to-peer media 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, Block-Oriented Probabilistic Verification (BOPV), an efficient protocol that 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. 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. Finally, simulation and wide-area experiments are conducted to evaluate the performance of our protocols.

1 Introduction

Consider the following media distribution system: a central server (say, Hollywood) first starts the streaming distribution of some media files. 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, unlike authentication for Multicast [3, 15, 13], the suppliers in this environment are not assumed to be trusted. During multicast, packets come from a trusted server. In a P2P system, packets signed by any arbitrary supplier may not be acceptable to other peers. Thus, the requesting peer needs a point of reference to verify the data it receives from the suppliers. 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 the peer 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 without causing significant delay. The system has several other issues such as how to ensure that the clients will not distribute their contents to their friends. These issues are out of scope of this paper. We focus on the data integrity verification in this paper.

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 first propose a Block-Oriented Proba-

[*]School of Information and Management Systems, University of California, Berkeley. Part of the work is done when the author was at Purdue. Ahsan Habib is supported in part by NSF-ITR 0085879 and NSF-ANI 0219110. Email: habib@sims.berkeley.edu

[†]Center for Education and Research in Information Assurance and Security (CERIAS) and Department of Computer Sciences, Purdue University, West Lafayette, Indiana. Dongyan Xu is supported in part by NSF-IIS 0209059, Mikhail Atallah is supported in part by NSF-EIA 9903545, NSF-ISS 0219560, and CERIAS, and Bharat Bhargava is supported in part by NSF-ANI 0219110 and CERIAS. Email: dxu,mja,bb@cs.purdue.edu

[‡]School of Information and Management Systems, University of California, Berkeley. John Chuang is supported in part by NSF-ITR 0085879. Email: chuang@sims.berkeley.edu

bilistic Verification (BOPV) protocol to verify data integrity efficiently. We show that probabilistic verification provides high assurance of data integrity and incurs low computation overhead. Then, we propose One Time Digest Protocol (OTDP) and Tree-based Forward Digest Protocol (TFDP) to 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.

We note that if the authentication process is not required, we achieve data verification without involving the Hollywood server in each session. However, a trusted party is required to provide a point of reference for each media file. The reference information will be posted on a set of distributed sites. This process is similar as posting the rendezvous point for an application layer multicast system, or obtaining information about a close by peer in a peer-to-peer system such as PASTRY [20]. When a peer wants to watch a movie, it downloads the trusted reference information and then proceeds with the proposed protocols. In this scenario, we like to promote the TFDP protocol because it requires to post only one signed hash as a point of reference. We also note that the proposed protocols are not limited for media streaming sessions. We impose the timing constraint to make it applicable for multimedia streaming. If this restriction is relaxed, the solutions still work. For example, data sharing in a P2P environment can use the proposed protocols.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 provides our protocols and discusses protocol parameter configuration in order to obtain desired level of security. Analysis and comparisons among different authentication schemes are presented in Section 4. Section 5 provides results from simulation and experiments from wide area setup of our prototype implementation. Finally, Section 6 concludes this paper.

2 Related Work

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 RSA digital signature [18]. A peer caches the packets as well as the signatures, when it watches a media file. The signatures along with the data will be provided to other peers upon request. The receiving peer can verify the digests using the server's public key. The main drawback is that the RSA signature verification incurs high computation overhead at the receiving peers.

Gennaro and Rohatgi [6] introduced techniques to sign off-line and online digital streams. This method is elegant, however, it does not tolerate packet loss and it has high communication overhead. Although one-time and fast signature schemes such as [11, 14, 17] can reduce computation and/or communication overhead, these signatures are only secure for a short period of time. Rohatgi [19] 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 [21] studied data authenticity and integrity for lossy multicast flows. They proposed Merkle'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 Merkle's tree. However, we significantly reduce the overhead by sending the digests of one subtree before sending any data.

Perrig *et al.* [15, 16] 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. Furthermore, if supplying peers generate keys and sign the digests like TESLA, it might not be acceptable to other receiving peers in a P2P streaming because the 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 explore this option to make our protocols robust against packet losses during streaming.

Park *et al.* [13] 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. 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. We show that our protocols incur much lower overhead than SAIDA, yet it achieves satisfactory loss resilience.

Horne *et al.* [9] 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 media streaming due to the unacceptable latency and overhead of verifying every single segment via the escrow server. Castro *et al.* [5] address different security issues in P2P network routing. They show that *self-certifying* data can help secure P2P routing. However, they do not address 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 media streaming.

3 Proposed Solution

To design the protocols, 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 a trusted authority for data integrity verification. The following models are considered to design the protocols.

Streaming Model. Let 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 whole streaming session. We assume a media file is divided into a set of M segments 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. We use block and segment interchangeably. A series of contiguous packets is referred to as a segment or block, and a series of segments is referred to as a group. Figure 1 shows this relationship.

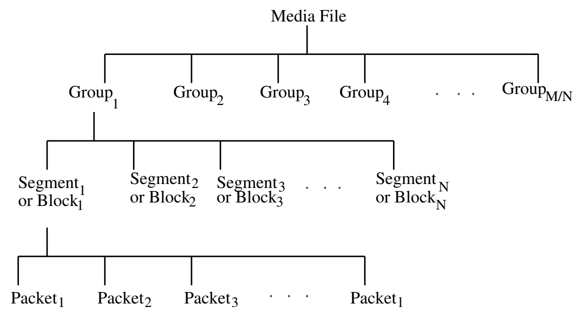


Figure 1: A media file is divided into Ml packets, l contiguous packets are labeled as a segment or block, N contiguous segments are called a group. Thus, the media file has total M segments and M/N groups.

An entire segment can come from one peer or a set of peers can contribute to stream one segment. Multiple peers participate to provide part of each segment to ensure that the receiver obtains the data at the estimated streaming rate. We define an active set of peers $\mathbb{P}^{act}$ who participate simultaneously at any time of the streaming session. The streaming protocol is responsible for the data assignment to the set $\mathbb{P}^{act}$. For details about the data rate assignment to each peer, readers are referred to [8]. The active set can be changed due to change in network dynamics. The receiver peer might not need to download the digests that are used for verification if it already has the digests from previous set of peers.

Incentive Model. An incentive mechanism is required in this system to motivate the peers to become suppliers. Otherwise, most of the peers will not cooperate as it is reported in [2]. The incentive model describes the utility a supplier can obtain for becoming a supplier. The peer calculates its cost for storing data and streaming. If the cost is less than the benefit it receives, a peer decides to cooperate. The details of the incentive model is out of scope of this paper. However, it plays an important role to act a supplier as an adversary, where the peer claims it has the data, however, it sends garbage data and tries to receive the benefit.

Adversary Model. In this model, any sender/supplier of the media file puts garbage data in any segment at any time 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 any media file, which foils our objective that a receiver should have the ability to verify the integrity of downloaded data. An adversary can intentionally drop some of its own packets or others packets to pretend that the network is congested.

3.1 Block-Oriented Probabilistic Verification (BOPV) Protocol

The Block-Oriented Probabilistic Verification (BOPV) Protocol uses the merit of taking hash of a block of packets instead of hashing one packet at a time to reduce communication overhead. The probabilistic verification further reduces the computation overhead by verifying selective segments instead of all segments. The probabilistic verification still provides high security. First, we discuss about the protocol, and then we analyze the pros and cons of it. The BOPV protocol, shown in Figure 2, runs as follows:

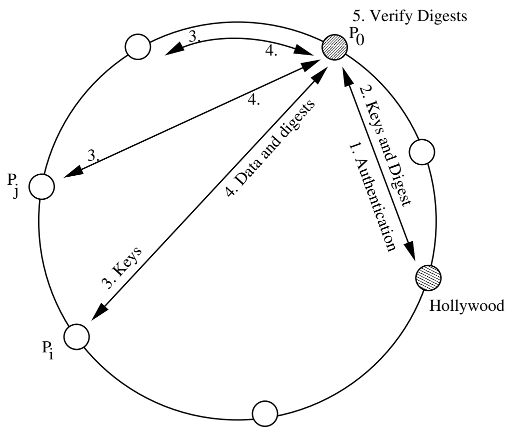


Figure 2: Steps of the BOPV protocol to receive data and verify digests by peer P_0.

- *Step 1:* Peer P_0 authenticates itself to the central Hollywood server. This is done using standard authentication process.

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

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

  where H_{ij} is the hash of packet j of segment i and h is a hash function. Keyed hash [10] can be used instead of Equation 1. However, we use this equation to show a simple example how this can be done.

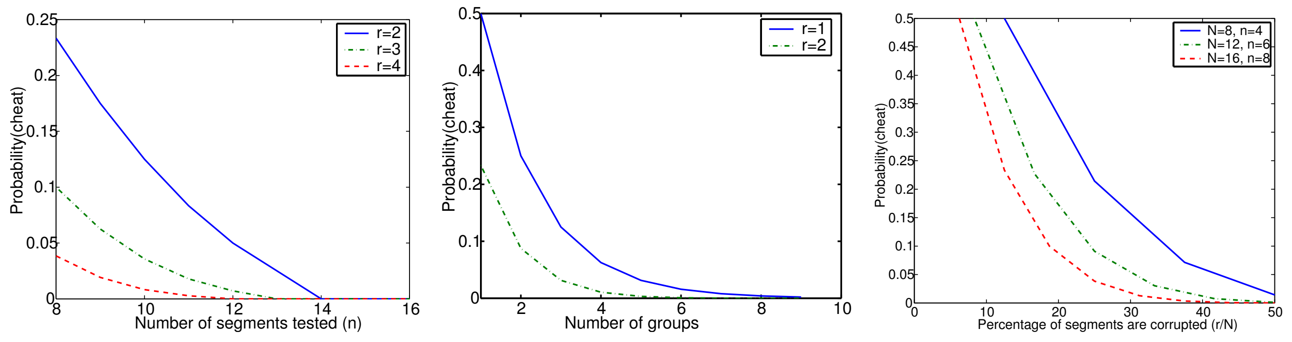
  The server groups the segments and provides P_0 only n digests out of the N segments in each group. These digests are used as a reference to verify the data downloaded from other peers. The communication is done securely. Step 1 and Step 2 can be done using public certificate or using Public and Private keys of P_0 and the server. These keys are different from the set of keys K.

- *Step 3:* P_0 gives each supplying peer one or more keys depending on how many segments the peers will provide. The main purpose of using these keys is to ensure that only a set of peers who are given keys can participate in the streaming process. An arbitrary adversary can not fool P_0 easily because it has to obtain the keys first. Without the keys the digests are not acceptable.

  To make the key distribution simple, one key is assigned for each segment. However, an improvement can be done by assigning one key for each peer. In that case, the caching strategy might enforce each peer to store a specified number of segments to ease the key management.

- *Step 4:* Each supplying peer uses the keys to generate digests and sends them to P_0 with the segments. The reason why the senders supply digests is twofold. First, P_0 can verify that the digest are coming from a peer that knows the keys. Second and the most important reason is that these digests are required in the verification process when some of the packets are lost.

- *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.



(a) (b) (c)

Figure 3: Successful probability of cheating (without getting caught) in different number of segments. N is the size of a group, n is the number of segments are verified out of N in probabilistic verification, and r is the number of segments an adversary wants to cheat in a group. (a) Increasing number of segments to verify (n) reduces successful cheating probability. (b) X-axis is the number of groups an adversary tries to cheat r segments. The cheating probability drops exponentially in multiple groups. (c) Successful cheating probability is very low when 30% or more segments are corrupted.

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 26 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 32 packets, Equation (1) reduces the volume of digests by 32 times to 0.835 MB. To further reduce the overhead, the server forms 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 the segments it gets digests from the server. Verifying 8 out of 16 segments will reduce the communication cost by 200%. Thus, the 26 MB communication overhead is finally reduced to 427 KB. If we further increase the segment size to 128, we can lower the overhead to 106 KB.

Probabilistic verification. It provides adjustable level of security and reduces computation overhead. In general, if a supplier 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 3 shows how the probability to cheat varies with number of segments verified by P_0 with the percentage of corrupted segments a malicious peer may try to send. Let N=16 and n=8. If a peer tampers with one segment, the chance of being detected this is only 50%. However, if a peer tampers with 4 segments, then more than 96% of the time 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.

The successful cheating probability drops exponentially when a peer attempts to cheat in multiple groups. Even the probability is only 0.5 to detect cheating when one segment is corrupted, the probability of successful cheating goes down to 0.002 when one segment is corrupted in 10 groups. Figure 3(b) shows that the probability of successful cheating is 0.0008 when two segments are corrupted in six groups. Thus, the computation (and sometimes communication) overhead can be reduced to half, and cheating can be detected with a very high probability at the same time.

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 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). Efficient multi-chained stream signature (EMSS) [16] 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.* [7], 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.

We provide an example to clarify how to use the digest scheme in a lossy environment. Let, $l = 5$ and $t = 2$. P_0 receives $h(K_i, H_{i1}, H_{i2}, H_{i3}, H_{i4}, H_{i5}, K_i)$ from the server. The sending 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 first and second packets are lost, P_0 verifies using hashes H_{i1} and H_{i2} provided by the sending peer with 4th and 5th packets, and computing hashes for rest of the packets. This example can tolerate up to two packet losses out of five packets. If three packets are lost, the peer has to reject of five packets of this segment because it can not verify the other two packets.

It means we allow t packet losses out of l packets of a segment and we perfectly checks the integrity of rest $l - t$ packets. This scheme increases the communication overhead during download by $\frac{t \times |H(.)|}{|p|}$, where the $|H(.)|$ is size of a hash function and $|p|$ is size of a packet. Using more packets per segments reduces the allowable packet loss keeping the threshold same. The threshold $t = 3$ for segment size 16 packets can tolerate almost 20% packet loss. It is possible to tune the system by changing the value of t and l.

Forward Error Correction (FEC). Park *et al.* [13] introduced erasure code to encode digests and signatures instead 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. However, if less than b packets are available to the receiver, the whole segment can not be verified. If b digests are available, P_0 first decodes the digests, and then 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 2, we propose the One Time Digest Protocol (OTDP). In this protocol, the server generates the digests as shown in Equation (1) for a set of keys $\mathbb{K}$. The server distributes all digests to different peers off-line. These are the peers who want to be a supplier in any streaming session. A suitable caching technique can be used to distribute these digests. 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 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.

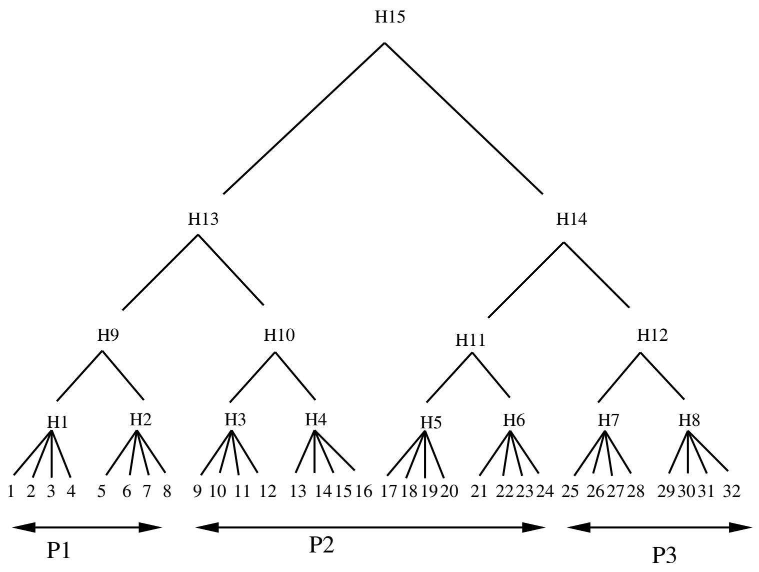


Figure 4: Tree structure of 32 packets that constitute 8 segments. P_1, P_2, and P_3 are members of an active set at different part of streaming session. P_1 is assigned to provide digests that are required to verify first two segments. P_2 provides necessary digests for next four digests and P_3 provides the rest. For example, P_1 sends H_1, H_2, H_{10}, and H_{14} to verify first two segments. P_2 and P_3 independently act in a similar fashion.

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

- *Step 5:* P_0 verifies each segment with appropriate keys.

In OTDP, P_0 downloads only a set of keys which is fairly small in 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 over lossy links. 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, these keys can not be used later; otherwise a peer can forge digest. However, the cache of media 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 digest download in *Step 1* of the BOPV protocol and to reuse the same digests over time, we propose Tree-based Forward Digest protocol. This protocol uses Merkle's tree, and is similar to *Tree-chaining* proposed by Wong and Lam [21] 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 Merkle's tree. Another difference is that our protocol creates one tree for a media file, instead of a separate tree for each segment.

In TFDP, a set of digests is downloaded first before downloading the corresponding segments. This periodical download of digests distributes the communication overhead over the whole streaming time and over all peers participated in the streaming. This protocol is mostly designed to stream a media file that is known beforehand.

Initially, the server generates the Merkle's signature tree for a media file. The leaves of the tree are packets of a segment. All non-leaf nodes of the tree represent digests of leaves of their corresponding subtrees. The server enforces a minimum number of segments to cache at each peer so that the overhead of sending extra digests is amortized over a group of segments. During a streaming session, N_{min} digests are downloaded before downloading the original segments. The parameter N_{min} is used to adjust the overhead to verify data integrity. A high value of N_{min} will reduce the overhead, however, it will delay the streaming session.

We provide a simplified example with 32 packets that are part of 8 segments. Let P_1 is assigned to provide the digests of first two segments, P_2 provides digests of next four, and P_3 provides the rest as shown in Figure 4. From previous section, we know that a segment is downloaded from a set of active 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 segments 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 segment by segment using H_1 and H_2. P_2 and P_3 act independently in a similar fashion.

Figure 4 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 segments that contain the packets. The digest of the segment is obtained by taking hash of all packets using Equation (1). The size of a segment needs to be chosen carefully to ensure that it does not introduce delay to collect all packets of a segment. All the segments (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 segment 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} segments (Theorem 1). The theorem also proves that it requires the lowest number of extra digests in the verification process when the tree is binary. Figure 5 shows that number of extra digests required to verify a group of N_{min} segments is minimized when $d = 2$. Higher value of N_{min} can reduce the required number of digests, however, it might increases delay to download extra digests before downloading data.

Theorem 1 *The TFDP requires* $(d-1)\left\lceil \log_d \frac{M}{N} \right\rceil$ *extra digests to verify a group of N segments out of total M segments using a d-ary tree with M leaves. Moreover, the number of extra digests required is minimum for* $d = 2$.

Proof: See Appendix. □

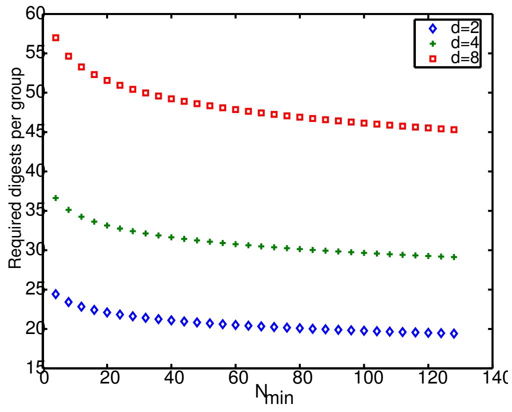


Figure 5: Number of digests required to verify the same number of segments using trees with different bases.

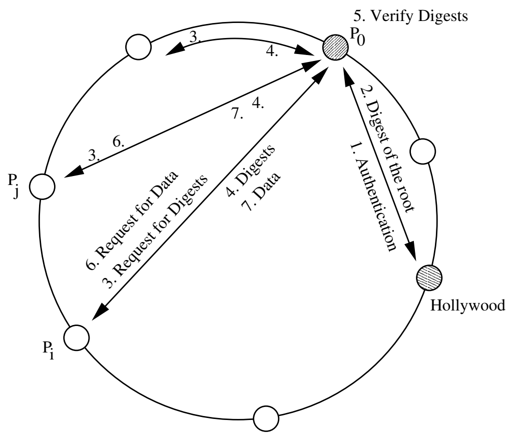


Figure 6: Steps of the TFDP to receive data and verify digests by peer P_0.

Now, we discuss the steps of the tree-based forward digest protocol. The TFDP runs as follows (Figure 6):

- *Step 1:* P_0 authenticates itself to the server.
- *Step 2:* The server provides P_0 the digest of the root of the Merkle's tree.
- *Step 3:* P_0 tells each peer P_i out of the active set $\mathbb{P}^{act}$ to forward the digests that are required to verify the N_{min} segments assigned to the active set.

- *Step 4:* 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. 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 segment of the N_{min} segments because the computed digest matches the digest of the root.
- *Step 6:* P_0 signals the peers that the digests are fine, and requests them to send data.
- *Step 7:* The peers send 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, 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. All digests are downloaded using TCP to ensure none of them is lost. The communication overhead is proportional to the height of the tree.

An important advantage of TFDP is that downloading digests is distributed over all peers. The TFDP requires to verify only one signature for the whole media file where as Tree chaining or SAIDA requires to verify one signature for each segment. 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.

4 Analysis and Comparison

In [13], the authors show that SAIDA performs better than EMSS [16] and augmented chaining [7] to tolerate bursty packet loss. We compare our three protocols with SAIDA and Tree chaining [21]. 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.

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

Tree chaining does not require to download any reference digest from the server, however, they require 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 segment, where l is the size of a segment in terms of packets. Each packet carries one 1024-bit signature. Thus, for each segment P_0 downloads $20l \log l + 128l$ bytes. All flavors of BOPV download one digest and one key from the server for each segment. The downloading overhead from the suppliers is high for multiple hashes. The OTDP downloads only keys from the server. Besides that BOPV and OTDP have similar communication overhead. The 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 segment. Again, digest of each segment is encoded with FEC. We define α, the overhead due to FEC as:

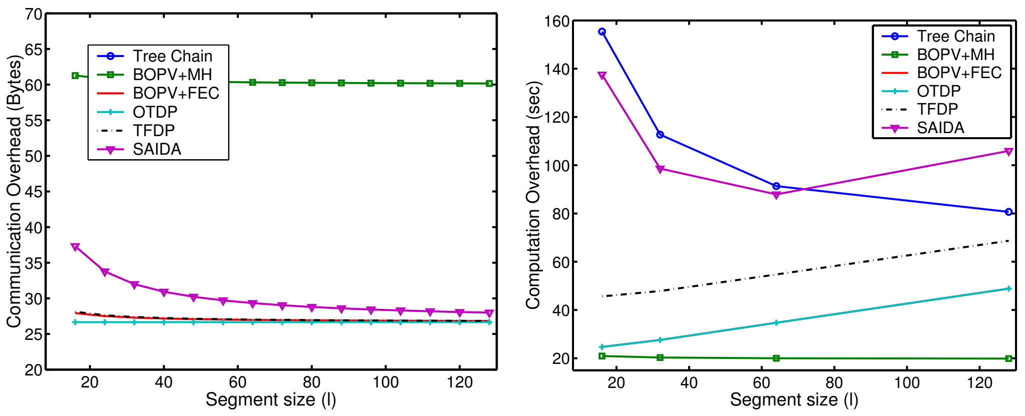
$$\alpha = \frac{\text{total packets sent per block}}{\text{total packets required to reconstruct the block}}. \tag{2}$$

Thus, the total communication overhead of TFDP is $20(l\alpha + 1 + \frac{1}{N_{min}} \log(\frac{M}{N_{min}}))$ bytes per segment. Only Tree chaining and SAIDA need to verify signatures. The SAIDA downloads one signature per segment, and it uses FEC. Thus, it requires $(20l + 128)\alpha$ bytes of overhead for each segment. All schemes verify the integrity of each segment based on the digest computation. Tree chaining and TFDP have similar number of digest computation because both of them use Merkle's tree. Others have almost same number of digest computation. Table 1 provides a summary for different protocols.

Communication Overhead Comparison. In tree chaining, each packet carries $20 \log l + 128$ bytes of extra information, which is significantly high. The SAIDA reduces the communication overhead by amortizing a

	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
OTDP	YES	KP_m	$20Ml\alpha$	$M(l+1)$	$Mv(l+1)$	—	—	M	probabilistic
TFDP	YES	20	$20Ml\alpha+20X$	$2Ml$	$2Mv(l+1)$	—	—	M	probabilistic
SAIDA	YES	0	$(20l+128)M\alpha$	$M(l+1)$	$M(l+1)$	M	M	M	deterministic

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 is the probability to verify a segment, α is defined in Equation 2, K is the size of a key, N_{min} is the minimum number of segment a peer caches, and $X = M + \frac{M}{N_{min}}\log(\frac{M}{N_{min}})$.



(a) Communication overhead

(b) Computation overhead

Figure 7: Overheads among Tree chaining, BOPV with multiple hashes and FEC, OTDP, TFDP, and SAIDA for movie *Matrix* of size, F=1.3 GB. The communication overhead is shown per packet and the computation overhead is for the entire file. The BOPV+FEC and OTDP have same computation overhead. Left figure does not show Tree chaining to highlight others.

signature over all packets of a segment. The BOPV with multiple hash has high communication overhead because each packet carries multiple digests of other packets to tolerate packet loss. Our experiments show that FEC requires less overhead than multiple hashes to tolerate same bursty losses. The OTDP has the lowest communication overhead, which is not surprising. This protocol uses one digest and some extra decode information for each packet, which is minimal for the verification process. The TFDP builds a tree of height $\log M$, which is larger than the height of the subtrees built by Tree chaining. However, the TFDP reduces the overhead by combining N_{min} segments together to make a group. Then, it downloads the necessary digests to verify all segments of the group. This reduces the height of the verification tree from $\log M$ to $\log \frac{M}{N_{min}}$. In addition, the TFDP does not use signature, which reduces the overhead significantly. The SAIDA acts similarly as TFDP, however, one signature per segment increases its overhead.

Figure 7 shows the analytical comparison of different protocols for the movie *Matrix*. Figure 7(a) shows that communication overhead can be reduced significantly if FEC is used to encode digests and signatures. The Tree chaining has extremely high communication overhead (208 B, for l=16, not shown in Figure 7). In

OTDP, each packet carries only 27B extra information. The TFDP and BOPV (both with MH and FEC) have more overhead than OTDP, however, less overhead than the SAIDA. The overhead of these protocols are close to each other when the segment size is large.

Computation Overhead. First, we describe the setup that is used to compare the computation overhead of all protocols. We use openSSL crypto library to calculate SHA-1 hash, RSA sign, and RSA verify. The Cauchy-based Reed-Solomon code [1] is used to encode digests in our protocols and in SAIDA. Tornado code [4] is a more efficient erasure code than Reed-Solomon code to encode/decode bulk size data. Our goal is to compare two protocols, and thus coding scheme does not affect our objective. 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.

The BOPV with multiple hashes requires only $M(l+1)$ hash computation. The Tree chaining does not have computation overhead for coding, however, it requires one signature for each segment, which increases the overhead significantly. The Tree chaining has M subtrees, and each tree evaluates $2l-1$ hash function, which is close to the value of TFDP. The TFDP requires to compute highest number of hash functions $2M(l+1)$, which is twice as high as the computation for SAIDA. Among all the schemes we compare, only Tree chaining and SAIDA uses digital signatures.

Figure 7(b) shows that the BOPV with multiple hashes has the lowest computation overhead. If FEC is used, the computation overhead increases with the segment 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. The overhead decreases with the segment size, however, its high communication overhead makes this solution hard to deploy in P2P streaming. Both TFDP and SAIDA uses coding to cope with packet losses. The SAIDA has higher computation overhead than TFDP because the SAIDA has to verify signature for each segment. Verifying a signature takes longer time than verifying a digest. The TFDP has higher computation overhead than BOPV (both with MH and FEC) and OTDP because TFDP sends few more digests for every N_{min} segments. We prefer TFDP over them because, unlike BOPV, TFDP reduces the initial communication overhead between receiving peer and the server and TFDP does not have the limitation of OTDP to use each digest only once. On the other hand, the BOPV and OTDP are less complex in nature.

5 Simulation and Implementation

We conduct simulations using the *ns-2* [12] 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. Like SAIDA, we use *Two-state Markov* loss model 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

$$V = \frac{1}{M}\sum_{i=1}^{M} \frac{\text{number of verifiable packets in segment } i}{\text{number of packets received in segment } i} \tag{3}$$

We compare SAIDA and OTDP in the simulation. The TFDP uses TCP to download digests, and the rest of the process is same as the OTDP regarding the computation of V. The outcome of the simulation is shown in Figure 8. The digests and signatures are encoded to tolerate 37.5% packet loss rate. We observe that due to burstiness, some segments have low packet verifiability. The reason why OTDP performs better is that SAIDA sends slightly more data than OTDP due to RSA signature for each segment.

We are developing *CollectCast*-based streaming system, called PROMISE [8] to provide high quality media streaming. It exploits network dynamics, quality of paths from sending peers to the receiver, and availability of the peers to make a set of peers available that can provide desired stream rate. Each of the proposed protocols can be plugged into PROMISE to verify data integrity in media streaming. We are evaluating our prototype by conducting experiments in PlanetLab test-bed. PlanetLab has hundreds of nodes in the USA, Europe, and Asia. We use a Berkeley node as a receiver. Nodes

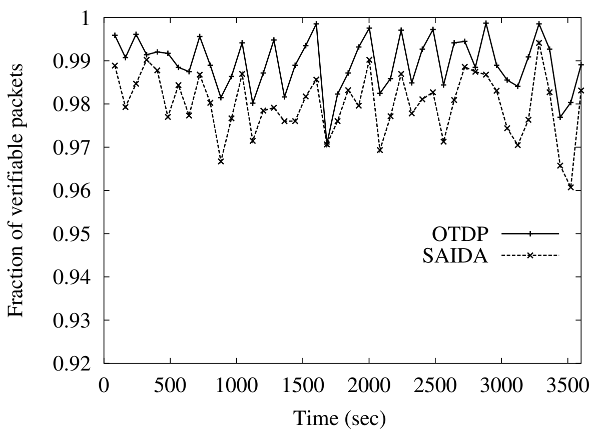


Figure 8: Packets verification probability at the receiver. More than 97% received packets are verified using OTDP. TFDP also has similar performance. The TFDP and OTDP vary in getting the reference data to verify integrity.

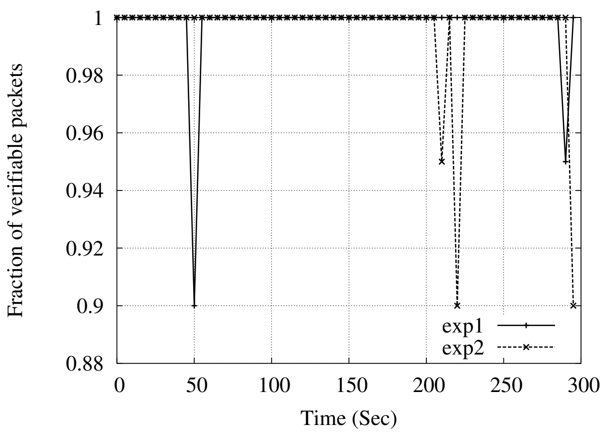


Figure 9: Packets verification probability at the receiver in the wide area setup.

from Purdue, MIT, Boston University, UK, and Taiwan are used as senders in this setup.

Figure 9 shows two PlanetLab experiments. Both can tolerate up to 20% packet loss due to FEC. If the loss rate is more than 20% for a certain time, an entire segment of the packets is not verifiable. Both experiments have similar loss rate. Exp 1 can verify almost all the packets throughout the experiment and thus the verification probability is 1.0 for 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 throw out couple of segments. Exp 2 experiences few more glitches than Exp 1. If the loss continues for a while, the supplier on the congested path is replaced with a better one. The wide area experiments shows that with FEC, the data integrity verification scheme achieves very high probability.

6 Conclusion

For P2P media streaming, data integrity verification is an important security issue. However, it receives less attention than other P2P security issues. In this paper, we propose efficient protocols to verify data integrity during P2P media streaming sessions. Our probabilistic packet verification protocol BOPV 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 significantly reduce the communication overhead to tolerate packet loss. 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 and implementation results show that a peer can verify 97% of packets even under a packet loss rate of 25%.

Acknowledgments

The authors would like to thank Mohamed Hefeeda for PROMISE prototype, and Maleq Khan for his help to prove Theorem 1.

References

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

[2] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), Oct. 2000.

[3] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Proc. of Eurocrypt '01*, Lecture Notes in Com-

puter Science, Vol. 2045, pages 437–452. Springer-Verlag, 2001.

[4] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *proc. ACM SIGCOMM*, Vancouver, Canada, Sept. 1998.

[5] 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.

[6] R. Gennaro and P. Rohatgi. How to sign digital streams. Technical report, IBM T. J. Watson research center, 1997.

[7] 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.

[8] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using CollectCast. Proc. ACM Multimedia '03, Nov. 2003.

[9] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *proc. ACM Electronic Commerce (EC '01)*, Oct. 2001.

[10] H. Krawcayk, M. Bellare, and R. Canetti. HMAC: keyed-hashing for message authentication, RFC 2104, 1997.

[11] L. Lamport. Constructing digital signatures from a one-way function. Technical report, SRI-CSL-98, SRI International Computer Science Laboratory, Oct. 1979.

[12] S. McCanne and S. Floyd. Network simulator ns-2. http://www.isi.edu/nsnam/ns/, 1997.

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

[14] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *proc. ACM Conference on Computer and Communications Security (CCS '01)*, pages 28–37, Philadelphia, PA, Nov 2001.

[15] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *proc. Network and Distributed System Security Symposium, (NDSS '01)*, San Diego, CA, Feb. 2001.

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

[17] 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.

[18] 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.

[19] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In *proc. ACM Conference on Computer and Communications Security (CCS '01)*, pages 93–100, Nov 1999.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Nov. 2001.

[21] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *proc. International Conference on Network Protocol (ICNP '98)*, Oct. 1998.

Appendix

Proof: Theorem 1. The height of a d-ary tree T with M leaves is $\log_d M$. The height of the subtree T_1 with

N leaves is $\log_d N$. To verify the root of T_1, we need $(d-1)$ digests from each internal node of the tree $T - T_1$. The height of the tree $T - T_1$ having $\frac{M}{N}$ leaves is $\log_d \frac{M}{N}$. Thus, the total extra digests required to verify N segments is $(d-1)\left\lceil \log_d \frac{M}{N} \right\rceil$.

To prove the second part, we show that higher value of d of a d-ary tree requires more digests to verify the same set of segments. It is sufficient to show that $d_1 \log_{d_1} Y > d_2 \log_{d_2} Y$, for $Y > 0$ and $d_1 > d_2 \geq 2$.

$\frac{p}{np} > \frac{2^p}{2^{np}}$, for some $p \geq 1$ and $n > 1$

$\Rightarrow \frac{\log_2 d_2}{\log_2 d_1} > \frac{d_2}{d_1}$, let $p = \log_2 d_2$ and $np = \log_2 d_1$.
As $n > 1 \Leftrightarrow np > p \Leftrightarrow d_1 > d_2$
$\Rightarrow \frac{d_1 \log_2 d_2}{d_2 \log_2 d_1} > 1$
$\Rightarrow \frac{d_1}{d_2} \frac{\log_{d_1} Y}{\log_{d_2} Y} > 1$, for any $Y > 0$
$\Rightarrow d_1 \log_{d_1} Y > d_2 \log_{d_2} Y$.

□