Loss-tolerant Real-time Content Integrity Validation for P2P Video Streaming

Fang Yu*, Vijay Gopalakrishnan†, K. K. Ramakrishnan† and David Lee*
*The Ohio State University †AT&T Labs – Research

***Abstract*—The use of peer-to-peer (P2P) mechanisms for content delivery is attractive to content and service providers alike. P2P data transfers offload the demand on servers and reduce the bandwidth requirements, with corresponding benefits of improved scalability and performance. This, however, poses interesting challenges in ensuring content integrity. Peers may be malicious and attempt to send corrupt and/or inappropriate content to disrupt the service. Consequently, service providers must provide clients with the capability to validate the integrity of content delivered from peers. This goal is particularly challenging in the context of streaming video because the content needs to be validated in real time. A practical solution must provide high integrity assurance while incurring low communication and computation overhead. In this paper, we present a packet-based validation approach for ensuring the integrity of data obtained from peers. Our proposed scheme randomly selects packets and validates their correctness. Through detailed experiments, we show that this mechanism is not only lightweight but is also able to detect content corruption with very high probability, thus protecting the viewing experience and the provider's content delivery service.**

I. Introduction

Peer-to-Peer (P2P) mechanisms are attractive alternatives for content delivery and are increasingly being used for video and audio streaming (e.g., BitTorrent [1], CoolStreaming [2], PPLive [3], Pando Networks [4]). The reason for using peers for video delivery is simple. Once a peer requests a video, it stores the video, and can serve it to other peers. Peers thus increase the capacity of the system at no cost to the content provider. [5] shows that peers can reduce the server load and bandwidth quite significantly. Additionally, peers can help in scaling the active library size. For example, peers who store copies of popular videos — videos they previously watched or were pre-loaded by the content provider — can serve requests of other peers. Meanwhile, servers can supplement the peers by serving unpopular and new videos.

However, the use of peer-to-peer mechanisms introduces new and interesting problems of content integrity and protection. Since users store the content locally, they have the ability to modify the content. Users may modify content out of malice (to bring disrepute to the service) or out of an intent to make a profit (by introducing ads or trying to take advantage of other revenue generation opportunities). Content may also get corrupted due to faulty hardware on the peers. Since the content or network service provider has little or no visibility to the data that gets transferred from the peers to the requester, the provider cannot validate the integrity of the content shared by a peer. Peers may also make unauthorized copies of the content or share the content over other P2P sharing sites. While preventing unauthorized use of content is an important challenge that providers have to solve, we do not explicitly address this problem in our paper. Solutions such as Digital Rights Management (DRM) [6] may be used to deal with these issues. Instead, our focus in this paper is on ensuring that the content obtained from peers, as part of the content delivery service, is valid.

The problem of validating content obtained across the network has been studied in detail [7]–[10]. The traditional approach is to compute a secure one-way hash of the downloaded content (e.g., MD5 or SHA-1) and compare the computed hash with that of the original content. This approach has been used effectively for software updates and file downloads. While hashing is applicable for our problem, it is challenging to directly adopt it in a peer-to-peer setting and especially with streaming videos. In a P2P setting, the client receiving the data has to be able to validate the integrity of the content by comparing it against some ground truth. This ground truth has to be communicated to the client securely by some means. Popular P2P protocols like BitTorrent, for example, use centralized trackers to transfer a torrent file that provides the client with hashes of pieces of the content. Others approaches include the use of Merkle trees [11], [12] for distributing the information to clients. However, they often assume bit-perfect delivery and furthermore approach the problem on a much coarser granularity than is applicable for real-time streaming.

With streaming videos, the content has to be verified in real time to detect corrupt data that causes impairments in the video, which is not acceptable, especially in a paid service. There have been proposals to achieve real-time validation by checking the data at fine-enough granularities (e.g., chunks of data [11], [13] or even at the level of every packet [14]–[16]). Some of these approaches, however, are designed to detect modifications to the packet in the network [9], [17] and cannot detect modifications by the source of the data. These approaches are also either expensive to use in practice and/or susceptible to packet losses especially when UDP is used as the transport. We seek a solution that is not only lightweight in terms of computation and communication costs, but is also resilient to occasional packet drops. For example, packet loss, to a limited extent, may be overcome by loss-concealment algorithms at the video player and therefore the validation technique also needs to be equally resilient.

In this paper, we address the problem of validating the integrity of videos obtained from peers by checking a random subset of packets at each client. We seek to strike a balance between ensuring that the content delivered to a client is authentic, and the overhead in computation, communication and storage resources used for such validation. We assume a setting where there is a service provider that has a large subscriber base requesting videos to be delivered on-demand over an IP network. The original content is available at trusted servers of the service provider. Peers complement the servers in delivering the content to requesting clients. It is in the interest of the service provider to ensure that the content received by its subscribers is valid. In order to do this, the server computes hashes for all the packets of the video upon ingesting the content from the content provider. This only needs to be performed once, as a pre-processing step. Rather than making a client check the hash of every packet it receives from a peer, the service provider determines, on a per-client basis, a random selected set of packets whose validity has to be checked. This is intended to keep the overhead at the client and the added communication overhead to send the hash values manageable. This per-client customized set of packet hashes are transferred to the client in a "playfile", which also specifies the content that needs to be fetched by the client. The client validates the packets as they arrive from a peer following this playfile. We use the CPM [18] system, a P2P video-on-demand implementation, as a working example to demonstrate that our approach not only works well, but is also practical.

We make the following contributions in this paper:

- Our approach seeks to strike a balance between overhead and ensuring the integrity of the content from peers. We show how we can take advantage of some of the properties of video encoding and of the deployment setting for designing a lightweight and practical approach.
- We show that our random packet-level detection mechanism is robust to a variety of strategies that a malicious peer may adopt. By controlling the number and type of packets checked, the provider can control the quality of service perceived by the viewer.
- Through detailed experiments using a real implementation, we show that we can reliably detect data corruption in real-time with high probability. We also show that the amount of overhead, in terms of computation and communication costs, both for the provider and the client, are minimal.

The rest of the paper is organized as follows. We present related work in Section II. We present a brief overview of CPM in Section III before presenting a detailed description of our approach in Section IV. We present experimental results in Section V and conclude in Section VI.

II. Related Work

We discuss work related to our paper in this section. We classify prior work into two broad categories: P2P distribution of data and videos; and validation of content integrity.

P2P content distribution: While P2P mechanisms have been used to build a variety of applications, mechanisms for distributing real-time content are relevant to this paper. PPLive [3], Pando Networks [4], Kontiki [19] are examples of commercial P2P-based streaming video distribution mechanisms. Similar work has also been done in the research community both in the realm of streaming and on-demand videos. Huang et al. [5] were the first to show the potential of P2P and peer-assisted prefetching in reducing server bandwidth costs. Protocols inspired by BitTorrent (e.g., [2], [20], [21]) divide content into fixed sized chunks; peers form a random mesh and use a randomized pull technique to distribute the video. The use of swarming has also been proposed in [22]–[26]. Annapureddy et al. [23] arrange the peers in a mesh topology and utilize a central directory server. Janardhan et al. [24] use a combination of BitTorrent and DHTs to locate chunks while Vratonjic et al [25] use Bullet [27] to construct an overlay mesh. Tewari et al. [28] proposed swarming-based live streaming for limited upstream bandwidths. BASS [26] is a near VoD service that employs a hybrid approach (video server combined with peer assists). Finally, CPM [18] uses a combination of peers and servers in a service provider environment to deliver VoD. In this paper, we apply our validation techniques to CPM, but believe that our scheme can easily be adopted to work with any of these existing mechanisms that rely on receiving blocks of data from peers.

Content integrity validation: The problem of content integrity validation has received quite a bit of attention. *Star Chaining* and *Tree Chaining* were proposed [7] for verifying multiple packets in a flow or a multicast stream. In Star Chaining, packets are grouped into blocks. Each block has a digest and a signature. The block digest is computed by first computing the digests (e.g., MD5 hash) of all the packets in the block. A block signature is then computed (e.g., using RSA) using the block digest. Packets are validated using *packet signatures* sent along with each packet. A packet signature consists of digests of all the packets in its block, the position of this packet in the block, and the block signature. When validating the packet, if the computed block signature matches the block signature carried in the packet signature, this packet is accepted as valid.

Tree Chaining aims to reduce the communication overhead of Star Chaining by taking advantage of a Merkle tree structure. Instead of including the digests of all other packets in the packet signature like Star Chaining, the packet signature contains the digest of each sibling node in the packet's path to the root. The digest of a node in the tree is computed by hashing the concatenation of the digests of its children nodes.

Star and Tree Chaining have been extended for content validation in a P2P environment. For example, in Block-Oriented Probabilistic Verification [11], the authors use keyed hashes (e.g., HMACs) to compute block signatures and make use of an authentication server to provide reference signatures. While techniques such FEC or sending redundant packet hashes are proposed to make the scheme robust to packet loss, this technique is not suitable for real-time validation. Tree-based Forward Digest Protocol is an enhancement in [11] based on tree chaining. Rather than creating a separate tree for each block as with tree chaining, a single tree is constructed for the entire media file. This way the server only needs to provide the client with the signature of the tree's root. Weatherspoon et al. [12] also leverage the Merkle tree structure for data integrity validation. However, their focus is more towards self-verifying reconstructed data instead of using it for real-time applications.

There have also been proposals to utilize peers to help validate content integrity. Michalakis et al. [8] propose an approach called *Repeat and Compare* for ensuring content integrity in untrusted P2P CDNs. Using cryptographically verifiable records and random sampling, they identify misbehaving replicas. Gkantsidis et al. [14] propose methods for validating content distributed using network coding. Servers compute digests of blocks in the original file. The authors show how clients can then derive digests, using the original digests from the server, to verify blocks that are generated on-the-fly by peers. In order to reduce the overhead of this verification process, they propose a cooperative scheme for validating content, and an individual peer does not validate every block. Peers actively cooperate with each other to detect corrupt blocks. Whenever a corrupt block is discovered, the peer sends alert messages to its neighbors. While our intent is similar, the approach proposed in [14] cannot be applied in our setting because the verification can only be done at a block granularity. The client has to download an entire block before it can verify it. Further, the cooperative approach to mitigate the overhead of verifying all the blocks may result in a corrupt block being used (displayed to the user) before an alert for that unverified block arrives at a client.

Li et al. [9] suggest incorporating the validation procedure into the processes for storing, forwarding, transcoding and transformation of the video. Like in [11], erasure codes are used for recovering lost authentication information. In contrast, our approach is oblivious to the specifics of the data that is being validated and can be used for P2P content distribution irrespective of the type of data. Also, we are loss-tolerant due to the fact that we validate at the packet level. Each packet is validated independent of the others. If a packet specified for validation is not received, this packet is ignored.

Finally, there has also been a lot of work on techniques for computing the validation information, such as the use of signatures [10], [29], keyed hash [11], [15]–[17], and homomorphic hash functions [30], [31]. In this paper, we assume a trusted server in the network maintained by the service provider. As a result, we believe using MD5 to generate signatures of packets is sufficient.

III. Overview of CPM

We use CPM as a representative system over which our scheme could be built. We present a brief overview in this section but refer the interested reader to [18] for a more detailed description of CPM. Figure 1 shows the entities in the CPM system. CPM approaches the problem of content distribution from a service provider's perspective by using trusted servers that maintain a master copy of each piece of content, and cooperative peers that assist in sharing and transferring content that they have cached. When a piece of content is requested, CPM identifies the best source of the content. This decision is made dynamically and depends on the cost of transferring the content from that source.

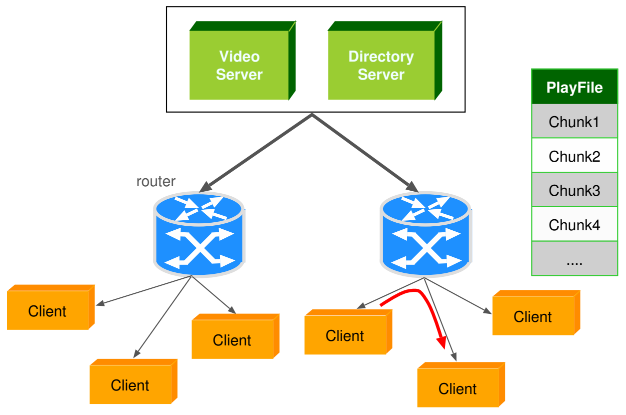


Fig. 1. Entities in CPM. CPM uses servers and peers to deliver chunks of data to requesting clients. Clients request chunks following a "playfile" provided to them upon request for the video.

A key aspect in CPM is that content is broken up and retrieved in relatively small *chunks*. Each chunk, with an associated unique identifier, is typically 30 seconds in length. These chunks serve as points where a new download decision can be made. In CPM, a client can switch from peer transfer to a server transfer and vice-versa for each new requested chunk, choosing the one that results in a lower transfer cost. Note that the use of chunks is similar in nature to that proposed in a number of recent approaches including [2], [20], [21].

Upon receiving a request for a video, the server provides the client with a playfile. The playfile describes the chunks in that video and the order in which they are to be played. The client dynamically determines and requests each chunk from the most appropriate source for the chunk. In practice, a client obtains a small number of chunks in parallel. Depending on the amount of time available, which we call *slack*, the client may choose to directly go to the server or try a peer transfer.

For policy reasons, CPM uses a *directory server* (DS) to aid the search mechanism. The DS's identity is programmed into each client. Each client informs the DS whenever it receives a new chunk or deletes a chunk from its local cache. When a client searches for a chunk, it simply asks the DS for the peers storing that chunk. The DS replies with a subset (configurable number) of peers, chosen uniformly at random, that have this chunk cached. The client determines if this set of peers can serve the content within the time available before the particular piece of content will be played. If not, it contacts the server for the chunk. Peers share any chunk they have cached. This includes chunks that were pushed to the peers through some pre-population mechanism or chunks that were downloaded for playback.

The server and the DS are trusted entities managed and protected by the service provider. While in principle the clients/peers are cooperating end-points, the system does not make an assumption that they are trusted entities. In fact, in this paper, we assume that the peers are end-systems that may be malicious as well as being prone to failures. As such, they may disrupt the operation of the content distribution service, be it intentionally or unintentionally.

IV. Validating Content From Peers

We now describe our approach for validating content obtained from peers. The goal of our approach is to validate the content that is received from a peer as quickly as possible, with minimal computational overhead on the client as well as the servers. Before we elaborate the specifics of our scheme, let us examine the various ways by which a malicious peer can disrupt the service.

A. Attack Model

The goal of a rational malicious peer that seeks to disrupt the service is to inflict the maximum damage while not being detected. Such attackers could attack the system either by delaying or not transferring portions of the requested chunk. As a result, the client receiving data from this peer would experience periodic glitches while watching the video resulting in a poor viewing experience. A peer could also attack the system by polluting the search mechanism. For instance it can claim to store chunks that it does not really have. This would result in a lot of clients coming to this peer for chunks it claims to store and in the process lose valuable time to download the chunk. Finally, the peer can attack the system by altering the chunks or transferring incorrect ones.

While each of these attacks is significant and needs to be addressed by a system employing P2P mechanisms, the focus of our work is primarily on detecting attacks where a malicious peer corrupts content. The following are some of the ways a peer could try to corrupt content:

Transfer incorrect chunks: The simplest attack by a malicious peer would be to transfer data that is different from what was requested. While this can be easily detected using the traditional hashing schemes, waiting for an entire chunk results in lost time and effort, which could in turn result in interruptions to the video.

Re-encode content: A determined peer could corrupt content by re-encoding it. The peer could, for example, re-encode the video with a different audio stream, down sample the video to a lower bitrate, or modify the video to play at a different speed. All of these could result in a poor viewing experience.

Insert content: A malicious peer could also insert content into the video. This could either be in the form of inappropriate content (to disrepute the service) or content such as advertisements (for profit).

Overlay content: Peers could overlay certain portions of the video with messages or pictures. For example, the peer could introduce a small bar at the bottom of the picture with advertisements or messages (as is commonly done broadcast TV), with unintended consequences for the provider.

Introduce glitches: Another form of attack, and one that is probably the hardest to detect, is one where the peer could replace packets uniformly at random with garbage content. Note, however, that it is not clear what the real-world manifestation of such corruption would be as it depends on the specifics of the video codec used, the bitrate of the video, the specific packets of specific frames dropped, etc.

In the rest of this section we describe our solution and show how our scheme is able to handle all of these attacks.

B. Content Validation by Verifying Integrity of Packets

Since each chunk of video is quite large ($\sim$30 secs in length), these chunks will be delivered using multiple (TCP or UDP) packets. To validate the streaming video content in real time, we propose to verify random sets of packets to ensure that these packets are not corrupt.

We believe that checking at the packet-level is the right granularity for verifying streaming video content for two reasons. Even small changes to the video can result in observable glitches. The real-time nature of the application means that verification needs to be performed as soon as the data is available and before it is used. This implies that we need to verify the data quickly and at very fine granularity before it gets used. Packet-level verification gives us a good balance between the cost of verifying the data and the flexibility to detect corruption quickly and take remedial action.

A valid question that comes up with random packet-level verification is: *Why sample randomly?* Ideally, one should be able to verify every single packet. Validating every packet not only guarantees against corruption going undetected but also enables early detection. The problem, however, is the overhead involved in performing validation. The client has to learn the "correct" digest of every packet so that it can compare the digest of the packet received against this correct digest. This can pose significant communication overheads. Next, the client has to perform the validation for every single packet received. This can pose significant computation overheads. Further, it may be unwarranted in settings where most peers are trustworthy. Hence, what is required is a mechanism that can be tuned to the specific setting. The service provider must be able to tune the system parameters such that the validation scheme can adapt to different customer types, content, device capabilities, prevailing system environment, etc. We believe that we can achieve this flexibility through random sampling of packets.

We split the problem of validating content into three complementary steps. In the first step, the server pre-computes the data that clients can use to validate received packets. In the second step, the server chooses a random set of packets that need to be validated and communicates this to the clients. The set of packets chosen is *unique* for each client. Since we are selecting at a packet level, this makes it hard for an attacker to guess which set of packets will be verified at a given client.

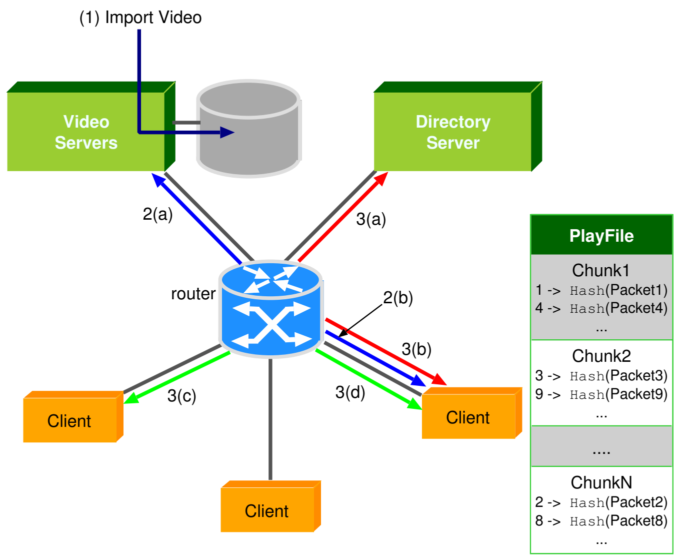


Fig. 2. Our 3-step process for validating content. The server first generates signatures for each packet. The client then obtains the signatures for the packets that it has to verify. Finally, the client compares the packets against their signatures as they are received.

While we could have allowed each client to independently pick the set of packets to verify, that would have resulted in the server having to communicate the hashes for all the packets to the client. Given the number of packets in typical videos, this would be very expensive. In the final step, clients perform the actual validation. Using the pre-computed data, the clients validate packets as and when they arrive from peers. As will soon be obvious, we leverage the trust that clients have of the service provider to eliminate complexity that the other proposals typically run into. The overall process is shown in Figure 2. In the rest of this section we describe in detail how each of these steps are performed.

Computing verification data: Recall our assumption that the provider is trustworthy and that clients trust the data received from the provider's server. We use these trusted servers as our source of validation data. In our scheme, servers compute the validation data when the video is ingested into the servers for distribution. Specifically, when a video is added into the server, the server processes the video and identifies the packets in the video. It then computes digests for each packet in the video. In this paper we use MD5 for generating the signatures, but could easily switch to other more secure hashing schemes, if required. The server computes these hashes once for each video and stores it along with the other meta-data associated with the video. We represent this as step (1) in Figure 2. Note that this approach could be performed on both raw or encrypted videos (or for that matter on any data) that the service provider may receive. The import mechanism is oblivious to the semantics of the content.

Communicating verification data to the client: An important step in the verification process is for the client to learn the valid signatures of packets that it wants to verify. Typically, this has been achieved through posting the signatures on a website (e.g., for file downloads) or though more sophisticated techniques such as Merkle trees [11] for distributed download schemes. Our approach, however, is to eliminate this complexity by making use of the trusted servers.

Clients contact the server when requesting a video, represented as step 2(a) in Figure 2. The server has been modified so that the playfile generated by the server not only contains information about the chunks in the video, but also includes information about the specific packets to be verified and their digests. It is important to note that (a) the server generates a unique playfile for each client, and (b) it is the server that dictates which packets are verified by each client. This policy ensures that the set of packets verified by each client is different and makes it hard for the attacker to predict which packets will be checked. Once a playfile is generated, it is transferred securely to the client (step 2(b) in Figure 2). This secure transfer is essential to prevent other peers from learning the set of packets that a client will check.

Verifying content integrity: Once a client receives the playfile, it executes the sequence of steps to download the chunks in the video. In general, the client may fetch a few chunks in parallel (like in CPM). For ease of exposition, however, we will assume that the client downloads one chunk at a time for playback. The client first tries to identify the set of peers that can serve the chunk. In order to do so, it contacts the directory server for peers capable of sharing that chunk. The directory server responds with a set of addresses of such peers. These are shown as steps 3(a) and 3(b) in Figure 2.

The client then attempts to download the chunk by contacting the peers identified in the directory server's response. It contacts these peers sequentially until it identifies a peer that can transfer the full chunk before it has to be played back (steps 3(c) and 3(d) in Figure 2). Once the download starts, the client uses the information specified in the playfile to start the verification process. For each packet of that chunk specified in the playfile, the client computes the digest of that packet when it arrives and compares it against the digest specified in the playfile. If the digests match, the client passes the packet. If there is a mismatch, the client may discard the packet and increment a counter that tracks the number of corrupt packets. When the counter hits a threshold number T of corrupt packets (T varies between 2 and 5 in our implementation), the client decides that the peer is transferring a corrupt chunk and aborts the transfer. It then restarts the process of identifying and downloading the chunk from a different peer. The client could reuse the list it got earlier from the directory server, or send a fresh query to the directory server for this purpose. In the worst case, the client would go back to the server and request the chunk to be delivered directly.

Observe that the threshold T is configurable. It can be set based on the perceived number of malicious clients in the system, data corruption rate observed in the network, etc. It may also be based on importance associated with that particular video. Moreover, our approach can trivially be extended to verify data from the server also, to handle man-in-the-middle attacks, etc. This is possible because the server provides information to verify packets in each chunk of the video (since any of the chunks in the video could be downloaded from peers).

C. Discussion

We make the following observations about the advantages and implications of using our approach.

- A malicious peer could impair the viewing experience by corrupting just a small number of packets. Since our approach is to check at the fine-grained level of each packet, we believe it can detect all of the attacks outlined in Section IV-A, as we show in Section V.
- It is important for an attacker to not be able to generate content whose hash matches that of the real content. For this, we rely on a one-way hash such as MD5 or SHA-1. If one wants even greater levels of security (because of vulnerabilities with MD5 or SHA-1), we could easily switch to even more secure hash functions.
- The server computes hashes using a predefined packet size (i.e., the maximum transmission unit or MTU). The actual packet size used while transferring between peers, however, may be different. We note that this is a non-issue if the validation process occurs after the transport protocol (such as RTP/UDP) reassembles the packet fragments to hand it to the application.
- Our approach allows the provider to retain flexibility over what and how many packets need to be verified. The provider can configure this differently for each video, and potentially for each customer, depending on factors such as video type (e.g., free vs. paid content), customer profile (e.g., Govt. vs. consumer), resolution of the playback device (e.g., high definition television vs. cell phone), etc. It also allows the provider to adjust to system factors such as video bitrates (e.g., high vs. standard definition), the number of malicious peers in the system, etc.
- The steps a client performs once it detects corruption could be a policy decision. The client may, for example, wait until the entire chunk is downloaded before handing the chunk to the player. It could also simply discard corrupt packets but keep a count of the number of corrupt packets detected. When the threshold is reached, the client could stop the transfer from the peer, and switch to an alternate source. Factors such as the type and real-time demands of the video could dictate the policy.
- The provider can take action once corruption is reported, based on configured policies. For example, clients may report the identity of the peer to the provider each time they receive a corrupt chunk from that peer. The provider, for its part, can maintain a "blacklist" of peers. As a first level of protection, the provider may then not give this blacklisted peer to clients as a source of content. The scheme however has to be engineered so that colluding peers clients cannot target specific peers by wrongly accusing the peer of giving corrupt content. This is part of our future work.

V. Results

A. Detection Probability

We make some brief observations on the probability of detecting corruption by an attacker and then demonstrate the effectiveness of our mechanism based on experiments with our prototype implementation.

1) Random validation: Habib et al [11] derive the probability of detecting corrupt packets when the attacker corrupts packets randomly. Integrity validation is also performed by sampling packets uniformly at random. The successful detection of corruption depends on the total number of packets, N (larger N translates to a longer stream of packets to validate), the probability of corruption (i.e., r packets corrupted out of N), and the sampling probability (i.e., n packets validated out of N).

As intuition would suggest, higher the validation probability (implying greater number of packets are checked) and higher the corruption probability, higher is the probability of detection. For a fixed corruption and detection probability, the probability of detecting corruption increases when the number of packets in a chunk, N increases. When validation is done at the chunk level, then for a small chunk size, one would end up needing to validate almost all the packets in the chunk to detect corrupted packets. This is especially true when the corruption probability is very low.

In an operational environment, because each chunk is obtained potentially from a different peer (e.g., with BitTorrent or CPM), the decision has to be made on a chunk-by-chunk basis. We examine the sampling probability required to get to a 0.999 detection probability for varying chunk sizes in Figure 3. We measure in terms of the length of a chunk of MPEG-4 video played out at 2 Mbps, and with each packet corresponding to a payload of 1472 bytes. As the chunk size goes from 1 second to 30 seconds, we observe that the sampling probability (i.e., number of packets to be validated) required to achieve a detection probability of 0.999 goes down very rapidly. The detection probability can be high even with small validation percentage and small corruption percentage, when the chunk size is 30 seconds as illustrated in Figure 3. This reinforces our design choice of having a reasonably large chunk size (e.g., 30 seconds).

2) Pseudo-random validation: We adapt our validation approach to video, where we choose to validate at least some packets of every frame in the video. This allows us to ensure that if all packets of a frame are altered, our scheme will detect such corruption. However, we do not want to trigger a reaction based on a random bit error at the source and hence attempt to validate at least 2 packets in each frame.

We can derive the detection probability as follows: Let N = total number of packets in the chunk; P = average number of packets in a frame; n = expected number of packets to be validated in a frame ($>= 2$); r = expected number of corrupted packets in a frame. Then

$$P(Detection) \;=\; 1 - \frac{N}{P}\frac{C(P-r,n)}{C(P,n)} \qquad (1)$$

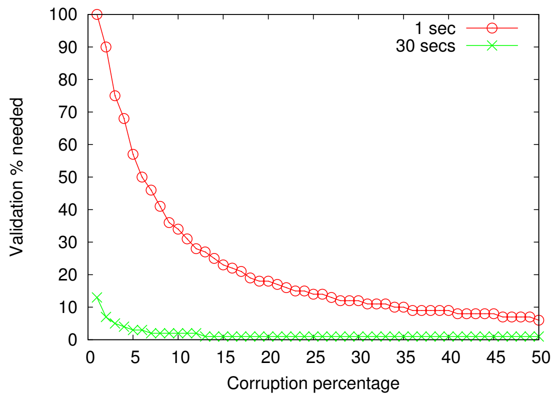


Fig. 3. Percentage of packets needs to be validated to provide 99.9% detection probability.

where $C(N, n)$ is the number of combinations to choose n out of a total N items.

We conjecture that this form of pseudo-random validation (since some packets of every frame are validated) is likely to be more robust against attacks that modify only specific frames (by replacing the frame, adding an overlay, etc).

B. Experimental Setup

We evaluated and compared random and pseudo-random validation through detailed experiments. The experiments were conducted by using a prototype of our detection mechanism. To achieve this, we extended the existing CPM server and client implementation to incorporate our scheme.

We extended the server such that it not only generates chunks of the video, but also computes hashes for all packets in each chunk. The server was also modified such that when it receives a request for a video, it generates a playfile that contains the chunks in the video, the sequence number of packets that the client needs to validate, and their corresponding hash values. The server generates a unique playfile for each requesting client. Finally, clients were modified to read this new playfile format, and to validate each of the packets marked for validation. Upon receiving that packet, a client generates a hash of that packet and compare the result with the value specified in the playfile.

We used both standard and high definition MPEG-4 videos to drive our experiments. In particular, we look at results with a 61 MB long standard definition video with a playout rate of 2.5 Mbps (approximately 20% of the packets are validated when selecting 2 packets/frame), and another 120 MB long HD video with a playout rate of 5.3 Mbps (approximately 10% of the packets validated when selecting 2 packets/frame). Our experiments were conducted on system with a 2.3 Ghz CPU, 2GB of memory and running GNU/Linux. We repeated each experiment 10 times. Unless otherwise specified, we plot the mean and 95% confidence interval from these 10 runs.

C. Simulating Corruption

In order to simulate a malicious peer, we modified a specific peer to corrupt packets that it transfers. We adopted a simple approach to corrupt content. Instead of actually introducing

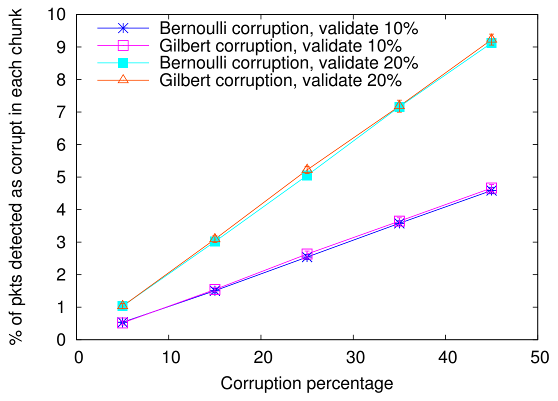


Fig. 4. Percentage of corrupt packets caught using random validation

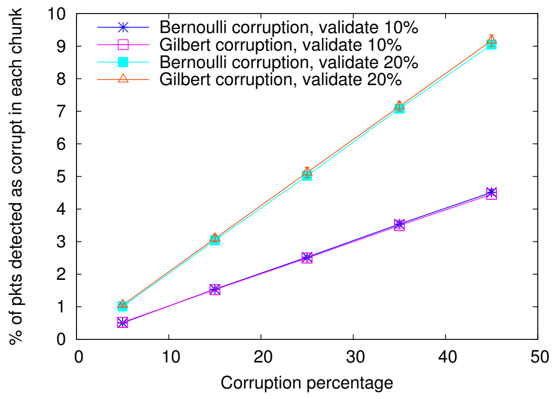


Fig. 5. Percentage of corrupt packets caught using pseudo-random validation.

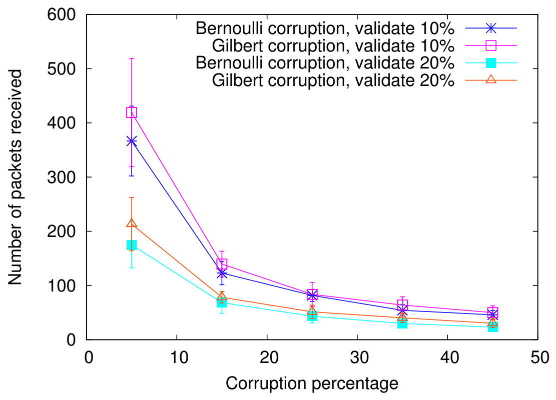
frames, or modifying frames, packets are corrupted by simply reversing the bytes in the payload.

Recall from Section IV-A, the different possible ways in which a malicious peer may corrupt content. Depending on the specific change to the video, one or more frames, and hence packets in the frame may be modified. The specific set of frames, and the number of packets that are corrupted depend on the exact change that the malicious peer performs. Since there does not exist any systematic mechanism to re-create the attack signal that a malicious peer's activity would result in, we resort to using distributions of corrupt packets that we believe capture the situations that could be observed in practice. In particular, the malicious peer uses either a *Bernoulli* model or a two state *Gilbert* process to pick the probability of corrupting the packet. While the Gilbert model is used to simulate a client that modifies packets in a bursty mode, the Bernoulli model is used to capture a client that corrupts packets randomly. By controlling the number of packets corrupted and using one of these models, we feel we can re-create the attacks specified in Section IV-A.

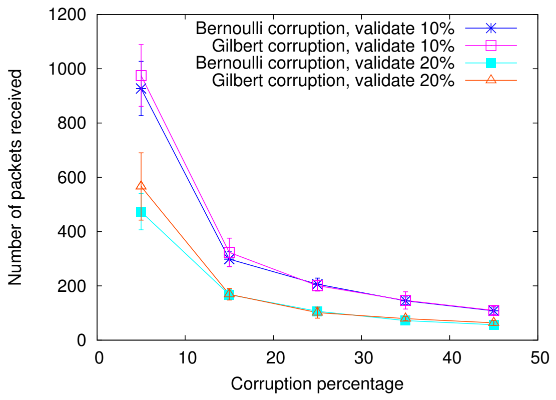
To summarize, when contacted by clients for data, this malicious peer modifies the payload of some of the packets, based on the corruption model employed, up to the specified percentage of corrupt packets.

D. Corrupted Packets Detected

In our first experiment we first verify how well our packet validation scheme performs. We run the experiment with both

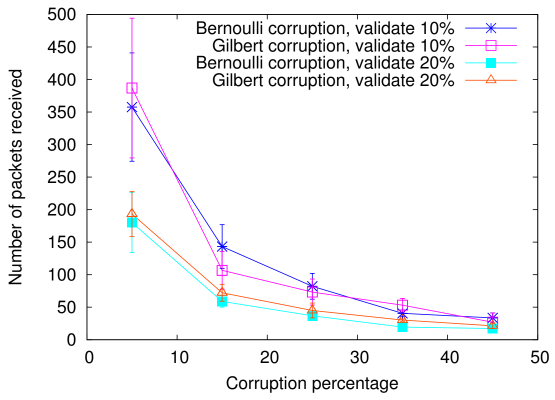


(a) When threshold is 2 corrupted packets

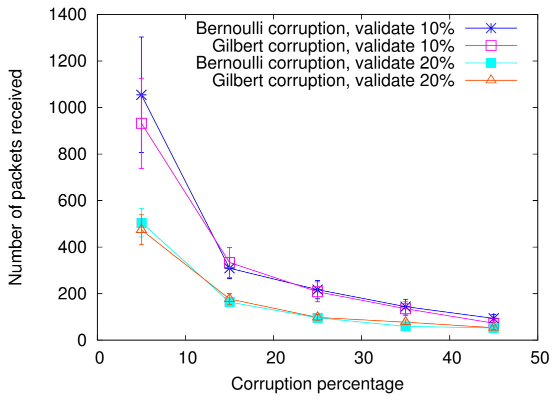


(b) When threshold is 5 corrupted packets

Fig. 6. Detection time (in terms of number of packets received) for random validation.



(a) When threshold is 2 corrupted packets



(b) When threshold is 5 corrupted packets

Fig. 7. Detection time (in terms of number of packets received) for pseudo-random validation.

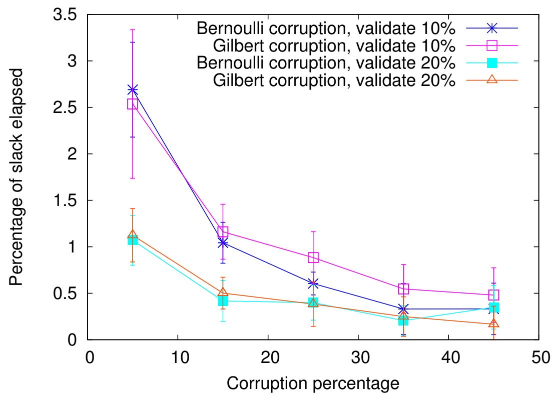
random and pseudo-random detection and plot the results in Figures 4 and 5 respectively. The X-axis shows the percentage of packets corrupted by the malicious peer, while the Y-axis plots the percentage of packets detected in each chunk by our scheme. As observed in the figures, the amount of packets detected as being corrupt grows linearly with increasing percentage of corruption for both random and pseudo-random detection. While this is not an unexpected result, the important takeaways from this result are that (a) our detection scheme works equally well for both Bernoulli and Gilbert models of corruption, and (b) pseudo-random performs as well as, if not better than, random in terms of detection.

E. Time to Detect Corrupt Content

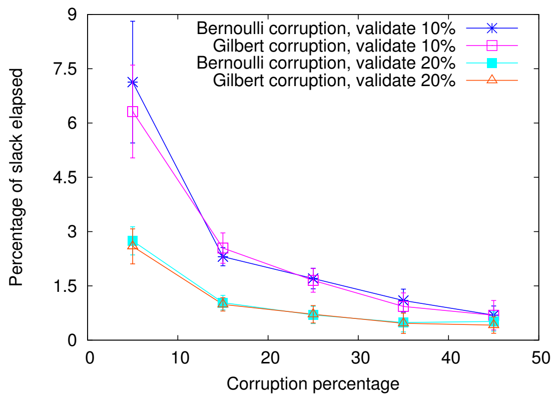
It is important to ensure that a client is able to detect corruption early so that it has time to recover and prevent the viewing experience from being affected. Specifically, in the context of CPM, when content can be retrieved from multiple peers, it is desirable to detect corrupt packets from a malicious peer early enough so that there is time to request the same chunk from a different source (either another peer or, in the worst case, the server). We measure how long it takes a client to detect a corrupt transfer.

Recall that we set a minimum threshold T for the number of packets to be detected as being corrupt before a client declares a peer to be malicious. We ran experiments for $T = \{2, 5\}$ and plot the results in Figures 6 and 7. The delay to detect the malicious peer reduces as the corruption probability goes up, as well as when the probability of sampling for validation goes up. As expected, when T increases, the delay in detection goes up significantly. When viewing it as a proportion of the packets in a chunk, we observe that even with relatively small corruption and validation probabilities we can detect a malicious peer prior to receiving half of the 30 second chunk (a chunk has approximately 5100 packets). The results are similar for both the random and pseudo-random validation.

From a practical standpoint, the time it takes to detect the corruption of content received from a malicious peer is important. In particular, the amount of time available before the particular chunk has to be displayed from the playout buffer is the time available for a client to recover the chunk from an alternate source. CPM's concept of "slack" remaining (i.e., the time available to download a chunk before it has to be played out) is measured for each of the techniques for different values of T. We plot the results of pseudo-random validation in Figure 8 (the results for random detection are similar). The plots show that only small percentage of the slack is consumed before a malicious peer is detected. This is encouraging because a large fraction of the slack is still available for a client to turn around and ask for the chunk from a different peer without disrupting the viewing experience.



(a) When threshold is 2 corrupted packets



(b) When threshold is 5 corrupted packets

Fig. 8. Detection delay (in terms of percentage of slack elapsed) for pseudo-random validation.

F. *Overhead of Our Scheme*

1) Computation: The server computes the hash for each packet once for a video when it is acquired from the content owner. Upon a request from a client, the server randomly or pseudo-randomly (based on configuration), picks a set of packets to be validated and includes the packet sequence numbers and the hash values for these packets in the playfile. There is no significant computational overhead at the server. The client does need to follow the playfile and validate the selected packets as the video is received.

We first measured the amount of CPU used both in terms of time and the number of cycles for computing the hash of a 1500 byte packet on a GNU/Linux system with 2 GHz CPU and 512 MB memory. It took approximately 6 microseconds to compute the MD5 hash of one packet using OpenSSL library. Even checking all the packets of a 1 second video (For 2 Mbps video rate, there are roughly 170 packets) will take about 1 millisecond, which is quite modest. In terms of the number of cycles, about 16256 CPU cycles are used for computing the hash of one packet. However, it is desirable to keep the amount of overhead small, so as to be able to perform this even in small, low-cost systems (such as set-top boxes and embedded systems including hand-held devices).

2) Communication: The communication overhead for validation comes from the hash values that must be communicated by the server in each playfile sent to clients. The number of bytes in the playfile depends on the validation percentage υ used by server. υ can be configured on a per video or a per client basis to provide different level of content assurance, based on the service provider's policy. Each selected packet for validation adds 16 bytes of hash value into the playfile. For example, with a 700MB video and $\upsilon = 10\%$, the extra size of the play file is a reasonable 0.76 MB.

3) Storage: The server has to maintain in persistent storage the hash values of all packets for each video. This results in an extra 16 bytes for every 1472 bytes of video data.

G. *Resilience to Packet Loss*

We ran experiments that measured the resilience of our scheme to random packet losses. Recall that in our scheme if a packet is to be verified, but is lost, then we simply ignore verifying that packet. In our experiment, we varied the loss probability from 1% to 5%. We measured the time to detect corruption when there are packet losses using the same parameters as our previous experiments. In all our experiments, packet loss did not affect our scheme: in the worst case, the average number of received packets increases by four extra packets than the case without loss.

This result showcases one of the strengths of our scheme. Compare this result to verifying at the chunk level. If the chunk level mechanism were oblivious to packet loss, then the client would not be able to distinguish between corruption and packet loss. Thus, it would not be able to compute the hash correctly and would wrongfully mark the peer as malicious. On the other hand, if the mechanism was packet-loss aware, then it would ignore validating this chunk. However, given the chunk size, even with low packet loss probabilities, there is a significant likelihood that each chunk would experience some packet loss. Hence, it would be impossible to validate any chunk, thereby negating the usefulness of the validation process.

H. *Detecting Malicious Peers through Cooperation*

So far we have shown that our mechanism allows a client to quickly detect corruption and switch to an alternate source for downloading content. In this experiment, we examine how quickly the malicious peer is caught when multiple peers receive content from it and when it attempts to transfer corrupt content to each of them. We set the corruption probability at 15%, the validation probability at 10% and vary the number of clients downloading content from this peer from 1 to 5. We measure the number of packets received, before detecting corruption, for both Bernoulli and Gilbert models and for $T = \{2, 5\}$. We plot the result in Figure 9.

As seen from the figure, when clients cooperate (report a corrupting peer to the directory server), we can detect malicious peers quickly. Also, as expected, beyond a point, the number of clients cooperating does not matter. The significance of this result is important: In a P2P environment, (a) cooperation can quickly prevent the spread of corrupt content, and (b) a client that detects corruption can help other peers downloading content from the same malicious peer by informing the DS about its detection.

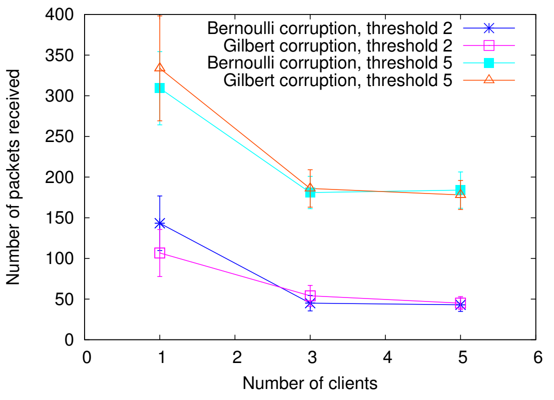


Fig. 9. Detection delay when multiple peers receive the content from the same malicious peer.

VI. Conclusion

Peer-to-Peer mechanisms hold much promise for distributing content. But their very nature makes these approaches susceptible to content manipulation and corruption by malicious entities. However, detecting corruption is challenging: the provider does not have visibility into the data transferred from peers, videos have to be validated in real time and packets may be lost in the transfer, etc.

We proposed a packet-level validation approach to detect content corruption by peers. Clients compute the hash of selected packets and compare it to the hash value for the corresponding valid packet provided by a trusted server. Our approach uses the insight that checking at the packet level gives us the right granularity and flexibility to detect corruption in real time, while keeping the overhead manageable. Since each packet is checked independent of the others, our approach is tolerant to packet loss. Our scheme allows service providers to control which and how many packets to verify. They can adjust the policy based on the importance of the video, the customer, and other factors. We present experimental results using a prototype implementation. We show our approach is quick enough so that a client has time to go to alternate sources when it receives corrupted content. Our approach is oblivious to different corruption rates, and the overhead of our approach in terms of communication, computation and storage at both the server and client is minimal.

The primary observation we make is that with the use of a trusted entity as a source of "ground truth", the problem of validating content in a peer-to-peer content distribution environment is considerably simplified. Our approach is practical and effective for a service provider to adopt and ensure that only valid content is displayed to a user with acceptably high probability.

References

[1] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Econ. of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[2] X. Zhang, J. Liu, B. Li, and Y.-S.P.Yum, "CoolStreaming/DONet : A data-driven overlay network for live media streaming," in *Proc. of IEEE INFOCOM*, March 2005.

[3] "PPLive," http://www.pplive.com.

[4] "Pando networks," http://www.pando.com.

[5] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proc. of ACM SIGCOMM*, 2007.

[6] K. Coyle, "The technology of rights: Digital rights management," http://www.kcoyle.net/drm_basics.pdf.

[7] C. K. Wong, S. Member, and S. S. Lam, "Digital signatures for flows and multicasts," in *Proc. of IEEE ICNP '98*, Oct 1998, pp. 502–513.

[8] N. Michalakis, R. Soul, and R. Grimm, "Ensuring content integrity for untrusted peer-to-peer content distribution networks," in *Proc. of the 4th USENIX Symposium on Networked Systems Design and Implementation*, Apr 2007, pp. 145–158.

[9] T. Li, Y. Wu, D. Ma, H. Zhu, and R. H. Deng, "Flexible verification of mpeg-4 stream in peer-to-peer cdn," in *Proc. of ICICS*, Oct 2004, pp. 79–91.

[10] R. Gennaro and P. Rohatgi, "How to sign digital streams," in *Proc. of CRYPTO*, 1997, pp. 180–197.

[11] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang, "Verifying data integrity in peer-to-peer media streaming," in *Proc. of MMCN*, Jan 2005, pp. 1–12.

[12] H. Weatherspoon, C. Wells, and J. D. Kubiatowicz, "Naming and integrity: Self-verifying data in peer-to-peer systems," in *Proc. of FuDiCo*, Jun 2002, pp. 91–94.

[13] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena, "The pollution attack in p2p live video streaming: Measurement results and defenses," in *Proc. of Peer-to-peer streaming and IP-TV Workshop*, Japan, 2007, pp. 323–328.

[14] C. Gkantsidis and P. R. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE Infocom'06*, Apr 2006.

[15] A. Perrig, R. Canetti, J. Tygar, and D. X. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *IEEE Symposium on Security and Privacy*, May 2000, pp. 56–73.

[16] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. of NDSS*, Feb 2001.

[17] H. Kim, "Secure scalable streaming for integrity verification of media data," in *Proc. of The 9th International Conference on Advanced Communication Technology*, Feb 2007, pp. 516–520.

[18] V. Gopalakrishnan, B. Bhattacharjee, K. K. Ramakrishnan, R. Jana, and D. Srivastava, "CPM: Adaptive video-on-demand with cooperative peer assists and multicast," http://www.research.att.com/∼kkrama/papers/cpm.pdf, August 2008.

[19] "Kontiki inc." http://www.kontiki.com.

[20] N. Magharei and R. Rejaie, "PRIME: Peer-to-peer Receiver-drIven MEsh-based Streaming," in *Proc. of IEEE INFOCOM 2007*, May 2007.

[21] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *IPTPS 2005*, Ithaca, NY. USA, Feb 2005.

[22] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing bittorrent for supporting streaming applications," in *Proc. of IEEE Global Internet Symposium*, Spain, April 2006.

[23] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is high-quality VoD feasible using p2p swarming?" in *Proc. of WWW*, May 2007.

[24] V. Janardhan and H. Schulzrinne, "Peer assisted vod for set-top box based ip network," in *Proc. of Peer-to-Peer Streaming and IP-TV Workshop*, Japan, August 2007.

[25] N. Vratonjic, P. Gupta, N. Knezevic, D. Kostic, and A. Rowstron, "Enabling dvd-like features in p2p video-on-demand systems," in *Proc. of Peer-to-Peer Streaming and IP-TV Workshop*, Japan, August 2007.

[26] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, "BASS: Bittorrent assisted streaming system for video-on-demand," in *7th IEEE Workshop on Multimedia Signal Processing*, 2005.

[27] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proc. of ACM SOSP*, October 2003.

[28] S. Tewari and L. Kleinrock, "Torrent assisted streaming system for video-on-demand," in *Proc. of IEEE NIME Workshop, CCNC*, Las Vegas, January 2007.

[29] D. Gavidia and M. van Steen, "Enforcing data integrity in very large ad hoc networks," in *Proc. of International Conference on Mobile Data Management*, 2007, pp. 77–85.

[30] M. N. Krohn, M. J. Freedman, and D. Mazi'eres, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *In Proc. of the IEEE Symposium on Security and Privacy*, May 2004, pp. 226–240.

[31] D. Luiz, G. Filho, P. Sérgio, L. M. Barreto, and E. Politécnica, "Demonstrating data possession and uncheatable data transfer. iacr eprint archive," Tech. Rep., 2006.