# Identifying Malicious Nodes in Network-Coding-Based Peer-to-Peer Streaming Networks

Qiyan Wang, Long Vu, Klara Nahrstedt, Himanshu Khurana
University of Illinois at Urbana-Champaign
Champaign, IL, 61801, United States

***Abstract*—Researchers show that network coding can greatly improve the quality of service in P2P live streaming systems (e.g., IPTV). However, network coding is vulnerable to pollution attacks where malicious nodes inject into the network bogus data blocks that will be combined with other legitimate blocks at downstream nodes, leading to incapability of decoding the original blocks and degradation of network performance. In this paper, we propose a novel approach to limiting pollution attacks by identifying malicious nodes. In our scheme, the malicious nodes can be rapidly identified and isolated, so that the system can quickly recover from pollution attacks. Our scheme can fully satisfy the requirements of live streaming systems, and achieves much higher efficiency than previous schemes. Each node in our scheme only needs to perform several hash computations for an incoming block, incurring very small computational latency in the range of several microseconds. The space overhead added to each block is only 20 bytes. The verification information given to each node is independent of the streaming content and thus does not need to be redistributed. The simulation results based on real PPLive channel overlays show that the process of identifying malicious nodes only takes a few seconds even in the presence of a large number of malicious nodes.**

## I. Introduction

Advent of multimedia technology and broadband surge lead to popularity of various online multimedia applications. One representative instance is live video streaming (e.g., IPTV), where a streaming server distributes live video streams to a large population of users over the Internet. The successes of several commercial peer-to-peer (P2P) streaming products, such as PPLive and SopCast, have demonstrated that P2P streaming is a promising solution to efficiently distributing live video streams at a large scale. Recently, researchers [2]–[4] found that *network coding* can greatly improve the quality of service in P2P live streaming systems with respective to high playback qualities, short buffering delays, minimal bandwidth costs, and resilience to peer dynamics.

Network coding has been initially shown to maximize the network throughput and robustness to link failures [5]–[7]. Unlike the traditional "store-and-forward" routing, network coding allows participating nodes in a network to code incoming data blocks (typically by linear combination), rather than simply forward them. However, the "combination" nature of network coding makes it vulnerable to *pollution attacks*, where malicious nodes inject into the network bogus blocks that will be combined with other legitimate blocks at downstream nodes and consequently corrupted blocks will rapidly spread over the network. As a result, the network performance will be substantially degraded due to the wasted bandwidth distributing corrupted blocks and the sink nodes will suffer from the incapability of decoding the original blocks. Thus, network coding cannot be safely applied to P2P streaming networks, unless the problem of pollution attacks is addressed.

The nature of live streaming implies two basic challenges on the defense scheme for protecting network coding from pollution attacks: (1) it should possess high computational efficiency to minimize computational delays; (2) the verification information given to the nodes for security checks on received data blocks (in a live video stream) should be independent of the content of the blocks. Additional requirements on the scheme include: (3) small communication overheads are introduced to save bandwidth; (4) the achievable network flow rate is independent of the power of malicious nodes (in terms of the number of links they can contaminate).

Several schemes dealing with network-coding pollution attacks have been proposed in the literature, but none of them can meet the above four requirements simultaneously. Generally, these schemes can be divided into two categories: *on-the-fly verification*, and *error correction*. The approaches in the first category allow each intermediate node to verify blocks on the fly. Typically, the previous schemes are based on public key crypto-systems with homomorphic properties, such as homomorphic hashing [8], [10] and homomorphic signatures [11], [12], which require expensive modular exponentiation computations at each hop, thus incurring substantial computational delays. To mitigate computational costs, researchers proposed probabilistic checking [9] or use null space properties of network coding [13]. However, the verification information in both schemes is derived from the blocks to be propagated, and must be repeatedly pre-distributed to all the nodes, leading to considerable delays and communication overheads. The second category – error correction – aims at correcting corrupted blocks at sink nodes by introducing a level of redundancy [16], [17]. Nevertheless, as a passive defense, error correction is applicable only when there are a limited number of corrupted blocks in the network, and the achievable flow rate is determined by the number of contaminated links.

Previous schemes only fight with corrupted blocks (either by the on-the-fly verification or by error correction), but do not limit the origin of the corrupted blocks – malicious nodes. Consequently, the malicious nodes can keep injecting bogus blocks to continuously degrade the network performance (e.g., by decreasing the network flow rate or wasting the nodes' CPU cycles on dealing with corrupted blocks).

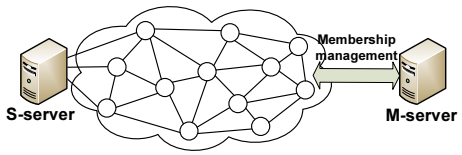


Fig. 1: Network model: mesh-structured P2P network.

In this paper, we propose a Malicious node Identification Scheme (MIS) that identifies and isolates malicious nodes, so that the pollution attack can cause harm to the network only for a short period of time and the subsequent streaming will no longer be influenced. MIS is block-based in that a malicious node can be identified repidly as long as it injects a single bogus block. To unambiguously identify malicious nodes, we design a novel and light-weight *non-repudiation transmission protocol* to ensure that any node that has injected a bogus block cannot deny its behavior and any malicious node cannot disparage any innocent node.

To the best of our knowledge, MIS is the first scheme to satisfy all the four requirements stated before. Each node in MIS only needs to perform several hash computations for an incoming block, incurring very small computational latency in the range of several microseconds, which is significantly smaller than most previous schemes. Besides, each block only carries a 20-byte evidence code, introducing much smaller communication overheads than any existing schemes. The verification information given to each node is independent of the streaming content and thus does not need to be redistributed. Furthermore, even if a large number of links are contaminated in the network, MIS is still effective and precisely identifies the malicious nodes. We simulate MIS based on real PPLive channel overlays with $1,600 \sim 4,000$ nodes, which are obtained in our previous work [19]. The simulation results show that the process of identifying malicious nodes only takes a few seconds even in the presence of a large number of malicious nodes. Although the main motivation of this work is to protect P2P live streaming, our scheme can also be applied to other network-coding-based applications, such as P2P large file distribution, as a deterrence of penalty to constrain pollution attacks.

This paper is organized as follows: Section II introduces the background and models. Section III gives an overview of MIS. Section IV presents the non-repudiation transmission protocol. Section V describes MIS in detail. Section VI gives the evaluation results and compares MIS with existing schemes. Section VIII concludes the paper.

## II. Background and Models

### A. Background of P2P Streaming with Network Coding

A P2P streaming network is a mesh-structured overlay network connecting dedicated servers and a potentially large number of nodes (or *peers*) over the Internet. As shown in Fig. 1, there are usually two types of servers in the system. One is the *membership server* (M-server) that manages the peer membership information, and the other is the *streaming server* (S-server) that supplies content over live video channels. The S-server may provide multiple channels simultaneously. Each channel forms a separate overlay, and each node belongs to one particular overlay at any time. In the remaining paper, we assume the nodes being considered belong to the same overlay (channel). There may be multiple duplicated streaming servers in the system, in which case we let "the S-server" mean "any one of all the S-servers".

When a new node $X$ joins the overlay, it first registers at the M-server, which provides $X$ a membership table that contains a list of peers in the overlay. Out of the membership table, some peers are selected by $X$ as its upstream neighbors, from which $X$ can download streaming content. $X$ may also request the S-server to serve it directly if none of the neighboring peers have the desired data. When some peers in the membership table are no longer alive, $X$ contacts the M-server to request more neighboring peers. Each live stream is divided into segments $\mathcal{S}_i$, $i = 1, 2, \cdots$, and each segment corresponds to a specific duration of playback (e.g., one second). To ensure smooth playback, each node maintains a *playback buffer* that consists of (tens to hundreds of) segments to be played. Each peer exchanges the information of availability of segments (referred to as *buffer map*) with its neighbors periodically.

When network coding is applied to the P2P streaming, each segment $\mathcal{S}_i$ is further divided into $m$ blocks $\mathbf{b}_{1,i}, \cdots, \mathbf{b}_{m,i}$, and each block $\mathbf{b}_{j,i}$ is subdivided into $d$ codewords $\mathbf{b}_{j,i} = (b_{1,j,i}, \cdots, b_{d,j,i})^\top$, $1 \le j \le m$. The segment $\mathcal{S}_i$ is considered as an $d \times m$ matrix of elements of the Galois field GF($n$) (e.g., $n = 256$), as shown below.

$$\mathcal{S}_i = (\mathbf{b}_{1,i}, \cdots, \mathbf{b}_{m,i}) = \begin{pmatrix} b_{1,1,i} & \cdots & b_{1,m,i} \\ \vdots & \ddots & \vdots \\ b_{d,1,i} & \cdots & b_{d,m,i} \end{pmatrix}$$

With network coding, both the S-server and peers perform encoding operations, which are applied only within a segment rather than across different segments. Whenever a peer or the S-server needs to forward a block to another peer, it produces a linear combination of all the blocks it currently stores.

In particular, when the S-server attempts to send a block $\mathbf{e}_{1,i}$ in $\mathcal{S}_i$ to peer $X$, it first picks $m$ random coefficients $(c_{1,1,i}, \cdots, c_{m,1,i})$ from GF($n$) (referred to as the *coefficient vector*). Then the S-server creates the coded block $\mathbf{e}_{1,i}$ by linearly combining the original blocks $\mathbf{b}_{1,i}, \cdots, \mathbf{b}_{m,i}$ with $(c_{1,1,i}, \cdots, c_{m,1,i})$, i.e., $\mathbf{e}_{1,i} = \sum_{j=1}^{m} c_{j,1,i}\mathbf{b}_{j,i} = (\sum_{j=1}^{m} c_{j,1,i}b_{1,j,i}, \cdots, \sum_{j=1}^{m} c_{j,1,i}b_{d,j,i})$. In other words, $\mathbf{e}_{1,i}$ is obtained by multiplying its coefficient vector $(c_{1,1,i}, \cdots, c_{m,1,i})$ with the $d \times m$ matrix $\mathcal{S}_i$ in GF($n$). The coefficient vector $(c_{1,1,i}, \cdots, c_{m,1,i})$ is appended to the coded block $\mathbf{e}_{1,i}$ and the augmented block is sent to peer $X$.

Assume peer $X$ has received $t$ coded blocks $\mathbf{e}_{1,i}, \cdots, \mathbf{e}_{t,i}$ in $\mathcal{S}_i$, either from the S-server or from other peers. When $X$ needs to transmit a block $\mathbf{e}_{t+1,i}$ in $\mathcal{S}_i$ to its downstream neighbor $Y$, it first picks $t$ random numbers $r_1, \cdots, r_t$ from GF($n$), and produces $\mathbf{e}_{t+1,i} = \sum_{k=1}^{t} r_k\mathbf{e}_{k,i}$. The coefficient vector $(c_{1,t+1,i}, \cdots, c_{m,t+1,i})$ of $\mathbf{e}_{t+1,i}$ (s.t., $\mathbf{e}_{t+1,i} = \sum_{j=1}^{m} c_{j,t+1,i}\mathbf{b}_{j,i}$) is computed by $c_{j,t+1,i} = \sum_{k=1}^{t} r_k c_{j,k,i}$,

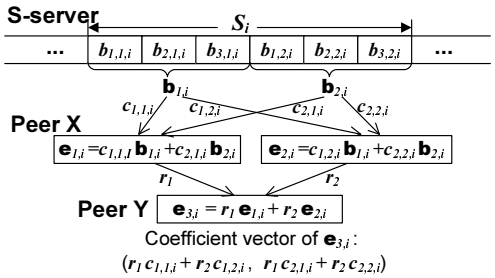


Fig. 2: An example to illustrate network coding in P2P streaming. Each segment consists of $m = 2$ blocks, and each block has $d = 3$ codewords. Peer $X$ receives two coded blocks $\mathbf{e}_{1,i}$, $\mathbf{e}_{2,i}$ in $\mathcal{S}_i$ from the S-server, and produces a new coded block $\mathbf{e}_{3,i}$ for peer $Y$.

$1 \le j \le m$. Then $\mathbf{e}_{t+1,i}$ together with $(c_{1,t+1,i}, \cdots, c_{m,t+1,i})$ is sent to $Y$. Fig. 2 gives a concrete example to illustrate the encoding operations of network coding.

A received coded block is first cached in the *receiving buffer*. A peer can reconstruct the original segment after accumulating $m$ coded blocks (within the segment) for which the associated coefficient vectors are linearly independent. The decoding process is similar to solving a system of linear equations. The decoded segment is cached in the playback buffer.

### B. Attack Model and Assumptions

We assume that a potentially large number of nodes in the overlay are malicious, but the majority of nodes are innocent. A malicious node could send any bogus blocks to any of its downstream neighbors, and eavesdrop, modify or simply drop any messages passing through it. A malicious node can exhibit these behaviors either alone or in collusion with other nodes. The main purpose of malicious nodes is to prevent innocent nodes from reconstructing the original blocks or to degrade network performance. We assume malicious nodes are "smart", and try to hide themselves or disparage innocent nodes.

We assume both the S-server and the M-server are trusted, and publicly known so that each node can contact them directly. We assume there exists a reliable PKI (Public Key Infrastructure) that enables each node to securely obtain the servers' public keys. These public keys are used by the servers to broadcast authenticated information (e.g., the result of identifying malicious nodes) to the overlay. Broadcast authentication is an important security primitive and has been extensively studied (e.g., [23], [24]), but it is orthogonal to this work.

## III. Overview of Our Scheme

In network-coding pollution attacks, malicious nodes send bogus blocks to their downstream neighbors. An innocent node that receives a bogus block is infected, and the blocks it produces with this bogus block are also corrupted and may further infect its downstream peers. The contaminated links and infected nodes form a connected component (or several components if multiple malicious nodes exist) in the network. Our goal is to track the origin of corrupted blocks.

To achieve this, we first propose an approach to detecting the existence of malicious nodes. We let each decoding node detect

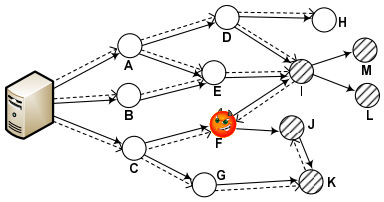


Fig. 3: An example of identifying malicious nodes in our scheme. The infected region consists of $I$, $J$, $K$, $L$, and $M$. Solid arrows denote the links of the streaming overlay, and dashed arrows denote the deliveries of the checksum. A malicious node (e.g., $F$) may block the dissemination of the checksum. With the checksum, $I$ (or $K$) discovers the corrupted block received from $F$ (or $J$), and $F$ (or $J$) is then reported to the servers. $I$ (or $K$) further forwards the checksum to its suspected neighbor $F$ (or $J$), but stop forwarding the checksum to its downstream neighbors $L$, $M$, since $I$ and the downstream peers $L$, $M$ share the same origin of the pollution attack (i.e., $F$).

corrupted blocks by checking if the decoding result matches the specific formats of video streams, and any node having an inconsistent decoding result will send an alert to the servers (the roles of S-server and M-server will be specified in Sec. V) to trigger the process of identifying malicious nodes. From the standpoint of the whole overlay, the existence of malicious nodes can be detected with very high probability.

After receiving an alert, the servers compute a checksum based on the original blocks, and disseminate it to the nodes using the streaming overlay. The checksum will help the nodes identify which neighbor has sent it a corrupted block. For example, in Fig. 3, with the checksum, node $I$ (or $K$) finds out that the block received from node $F$ (or $J$) is corrupted. Note that, at this point, one cannot confirm that the discovered nodes ($F$, $J$) are malicious, since they may be innocent and receive corrupted blocks from their upstream neighbors. For instance, $J$ is an innocent node and the corrupted block it produces is due to the bogus block it has received from $F$; whereas, node $F$ discovered by $I$ is indeed a malicious node. To this end, the discovered nodes ($F$, $J$) are temporarily treated as *suspicious nodes*, and are reported (by $I$, $K$) to the servers. Then, the reporting nodes ($I$, $K$) further forward the checksum to their suspected upstream neighbors ($I$, $K$), as shown in Fig. 3.

If a suspected node is truely innocent (e.g., $J$), then with the received checksum it will identify at least one corrupted block it has received from its upstream neighbors (i.e., $F$), and correspondingly it will report its suspected neighbors ($F$) to the servers. On the contrary, if a suspected node is malicious (e.g., $F$), it cannot find a suspicious neighbor that sent it a corrupted block. Therefore, we let the servers judge a suspicious node based on whether it can report another suspicious node.

The correctness of the above process of identifying malicious nodes relies on the condition that no one can lie when reporting a suspicious node. To be concrete, any malicious node cannot disparage an innocent node that does not send a corrupted block, or cannot deny having sent a corrupted block (when being suspected). For example, $F$ cannot disparate $C$, or deny having sent a bogus block to $I$.

One way to achieve these requirements is to let each node

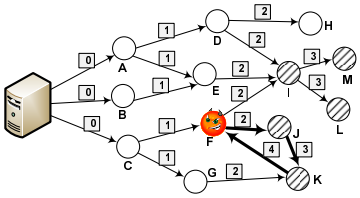


Fig. 4: An example of the infected cycles (bold arrows) in mesh-structured P2P streaming networks. The squares are the transmitted blocks, and the numbers denote the sequence numbers of the blocks.

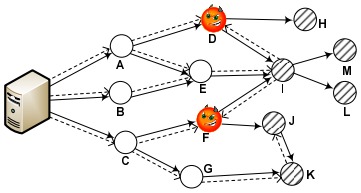


Fig. 5: An example of handling multiple malicious nodes. The infected region due to $D$ consists of $H$, $I$, and $L$, while $F$'s infected region contains $I$, $J$, $K$, and $L$.

sign the block it sends out using a public-key signature scheme, such as RSA. Then, the signature associated with the block can be used as the evidence by the reporting node to demonstrate that the reported suspicious node has really sent this block (given that it is infeasible to forge a public-key signature). However, this approach requires applying public-key signature on each transmitted block, which introduces substantial computational delays due to the expensive signature generation and verification. Alternatively, we design a light-weight *non-repudiation transmission protocol* (described in Section IV) based on efficient one-way hash functions, which can satisfy the above requirements with significantly higher efficiency in terms of both computational costs and communication overheads.

Note that in mesh-structured P2P networks, cycles possible exist. For example, in Fig. 4, $F$, $J$, and $K$ form an infected cycle in which each link is contaminated. In this case, since $F$ receives a corrupted block from $K$ (although the corruption of this block is caused by the bogus block injected by $F$), $F$ can report $K$ as a suspicious node to the servers, and consequently the servers cannot tell which node in the infected cycle is the origin of the corrupted blocks. To address this problem, we let each node append to the block it produces (say $\mathbf{e}_{t+1,i}$ in $\mathcal{S}_i$) a sequence number (denoted by $Seq(\mathbf{e}_{t+1,i})$), which is set as the maximum sequence number of all the received blocks $(\mathbf{e}_{1,i}, \cdots, \mathbf{e}_{t,i})$ in this segment plus one, i.e., $Seq(\mathbf{e}_{t+1,i}) = \max_{1\leq k\leq t}\{Seq(\mathbf{e}_{k,i})\} + 1$ (see Fig. 4), and the evidence associated with $\mathbf{e}_{t+1,i}$ is computed based on $Seq(\mathbf{e}_{t+1,i})$ and the content of $\mathbf{e}_{t+1,i}$. Then the node that has sent a block with the smallest sequence number among all nodes in the infected cycle is judged as the malicious node.

Our scheme can elegantly handle multiple malicious nodes, each of which has a corresponding infected region. As long as a node in the infected region (caused by a particular malicious node) receives the checksum, it can initiate the process of tracking the pollution origin, and at the end of the tracking process the malicious node of this infected region will be discovered. If a node belongs to multiple infected regions (e.g., node $I$ in Fig. 5 is in the overlap of the infected regions of both $D$ and $F$), it can help identify all the malicious nodes ($D$, $F$) of these infected regions. In addition, our scheme is resistent to collusion attacks, where multiple malicious nodes collude to hide themselves or disparage innocent nodes. We will analyze this in Sec. V.

*Remarks.* One major reason for us to seek a solution that identifies malicious nodes rather than deals with corrupted blocks is to minimize the overheads in the defense scheme, since high efficiency is a crucial requirement for live streaming systems. We observe that due to the "linear combination" nature of network coding, traditional security primitives with high efficiency (such as one-way hash functions) cannot be directly applied to network coding, and not surprisingly, the schemes [8]–[13] designed to meet the linear combination property inevitably suffer from low efficiency. Our scheme is targeting malicious nodes, instead of tackling linearly combinable blocks, and thus we can utilize highly efficient security primitives to construct our scheme, thereby achieving much higher efficiency.

Our scheme is not based on statistical algorithms, which require monitoring the traffic or the behaviors of nodes for a certain period of time before identifying the malicious nodes. Instead, our approach is deterministic and block-based. It can precisely and rapidly identify malicious nodes as long as they inject a single bogus block. This ensures that the system can quickly recover from pollution attacks.

## IV. Non-Repudiation Transmission Protocol

As we discussed before, the correctness of MIS is based on the condition that no node can lie when reporting a suspicious node that has sent a corrupted block, and an evidence associated with the corrupted block is necessary to demonstrate to the servers that the reported node has really sent the block. We design a non-repudiation transmission protocol to achieve this. We let $X$ be the suspicious node, and $Y$ be the reporting node. Let $\mathbf{e}$ denote the block[1] that $X$ transmits to $Y$ , $\Phi(\mathbf{e})$ denote the evidence (referred to as *evidence code*) associated with $\mathbf{e}$, and $Seq(\mathbf{e})$ denote the sequence number of $\mathbf{e}$ (used for handling infected cycles as analyzed in Sec. III). More notations used in this protocol are listed in Table I.

*Overview.* In the non-repudiation transmission protocol, before $X$ transmits $\mathbf{e}$ to $Y$, it generates an evidence code $\Phi(\mathbf{e})$ based on $\mathbf{e}$ and $Seq(\mathbf{e})$, and then sends $\mathbf{e}|Seq(\mathbf{e})|\Phi(\mathbf{e})$ to $Y$. For security concerns, $\Phi(\mathbf{e})$ is producible *only* to $X$, and verifiable to both $Y$ and the M-server (the role of the S-server will be described in Sec. V). The receiving node $Y$ checks whether $\Phi(\mathbf{e})$ matches $\mathbf{e}, Seq(\mathbf{e})$. If not, $\mathbf{e}$ is discarded. When $Y$ discovers that $\mathbf{e}$ is a corrupted block (after receiving the checksum disseminated by the servers), $Y$ sends a report $ID_X|\mathbf{e}|Seq(\mathbf{e})|\Phi(\mathbf{e})|ID_Y$ to the M-server. The M-server judges if $\mathbf{e}$ is sent by $X$, by checking whether $\Phi(\mathbf{e})$ is consistent

[1]Here we only consider one particular block $\mathbf{e}$ without specifying which segment $\mathbf{e}$ belongs to, so we omit the subscript of $\mathbf{e}$ as we used in Sec. II.

TABLE I: Notations

| | |
|---|---|
| $\Phi(\mathbf{e})$ | The evidence code associated with the block $\mathbf{e}$ |
| $Seq(\mathbf{e})$ | The sequence number of the block $\mathbf{e}$ |
| $\mid$ | Concatenation of two binary strings |
| $ID_X$ | The public identifier of $X$ (e.g., $X$'s IP address) |
| $scrt_X$ | The secret that node $X$ registers at the M-server |
| $\Upsilon(scrt_X, scrt_Y)$ | Secret elements derived from $scrt_X$ given to $Y$ |
| $\overline{\Upsilon}(scrt_X, scrt_Y)$ | Secret elements derived from $scrt_X$ unknown to $Y$ |
| $\lambda$ | # of values in an evidence code (e.g., $\lambda = 20$) |
| $\pi$ | Length of each value in evid. codes (e.g., $\pi = 8$ bits) |
| $\delta$ | # of secret elemts. in $\Upsilon(scrt_X, scrt_Y)$ (e.g., $\delta = 10$) |
| $\mathcal{F}(\cdot)$ | A pseudo-random function that maps the input into a $\delta$-element subset of $\{1, \cdots, \lambda\}$, $\delta < \lambda$ |
| $\mathcal{H}(\cdot)$ | A secure one-way hash function (e.g., SHA-256) |
| $trunc_\pi(\cdot)$ | A func. that truncates the input into leftmost $\pi$ bits |
| $\theta$ | The threshold used for verifying reports (e.g., $\theta = 4$) |

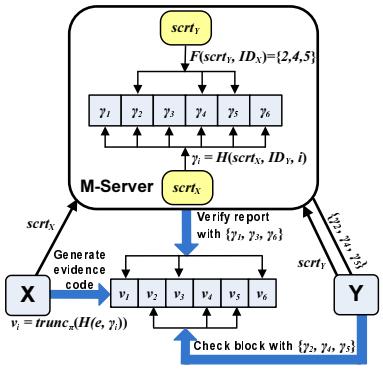


Fig. 6: An illustration of the non-repudiation transmission protocol with $\lambda = 6$ and $\delta = 3$.

with $\mathbf{e}, Seq(\mathbf{e})$. If not, the report is ignored. The probability that $Y$ can cheat the M-server with a block $\mathbf{e}'$ not sent by $X$ or that $X$ can cheat $Y$ with an evidence code $\Phi(\mathbf{e})'$ inconsistent with $\mathbf{e}, Seq(\mathbf{e})$ is negligibly small.

*Protocol description.* Now we describe the non-repudiation transmission protocol. We let each of $X$ and $Y$ initially register a secret (denoted by $scrt_X, scrt_Y$) with the M-server, respectively. $X$ will use $scrt_X$ to produce $\Phi(\mathbf{e})$. The M-server provides *partial information* of $scrt_X$ (denoted as $\Upsilon(scrt_X, scrt_Y)$) to $Y$ according to the information of $scrt_Y$. $\Upsilon(scrt_X, scrt_Y)$ can help $Y$ verify $\Phi(\mathbf{e})$.

The generation of $\Upsilon(scrt_X, scrt_Y)$ is as follows. First, the M-server maps $scrt_Y$ to a $\delta$-element subset $\mathcal{F}(scrt_Y, ID_X)$ of $\{1, \cdots, \lambda\}$, $\delta < \lambda$. For example, when $\lambda = 6$, $\delta = 3$, $\mathcal{F}(scrt_Y, ID_X)$ could be $\{2, 4, 5\}$. Second, the M-server derives $\lambda$ secret elements from $scrt_X$ by computing $\gamma_i = \mathcal{H}(scrt_X, ID_Y, i)$, $1 \le i \le \lambda$. Third, the M-server initializes $\Upsilon(scrt_X, scrt_Y)$ as an empty set, and then for each $i^* \in \mathcal{F}(scrt_Y, ID_X)$ the M-server adds $\gamma_{i^*}$ into $\Upsilon(scrt_X, scrt_Y)$. The finally obtained $\Upsilon(scrt_X, scrt_Y)$ is given to $Y$. Note that from $\Upsilon(scrt_X, scrt_Y)$, $Y$ cannot learn any information about $scrt_X$ or the secret elements that are derived from $scrt_X$ but not in $\Upsilon(scrt_X, scrt_Y)$ (i.e., $\{\gamma_j : j \in \{1, \cdots, \lambda\}/\mathcal{F}(scrt_Y, ID_X)\}$, denoted by $\overline{\Upsilon}(scrt_X, scrt_Y)$).

The evidence code $\Phi(\mathbf{e})$ consists of $\lambda$ values $\{v_1, \cdots, v_\lambda\}$, each of which is computed by $v_i = trunc_\pi(\mathcal{H}(\mathbf{e}|Seq(\mathbf{e}), \gamma_i))$, $1 \le i \le \lambda$. Once receiving $\mathbf{e}|Seq(\mathbf{e})|\Phi(\mathbf{e})$ from $X$, $Y$ verifies the validity of $\Phi(\mathbf{e})$ by checking if $v_{i^*}$ is equal to $trunc_\pi(\mathcal{H}(\mathbf{e}|Seq(\mathbf{e}), \gamma_{i^*}))$ for all $\gamma_{i^*}$'s in $\Upsilon(scrt_X, scrt_Y)$. Only a block with a valid evidence code (all the checked values $v_{i^*}$'s are consistent with $\mathbf{e}, Seq(\mathbf{e})$ and $k_{i^*}$'s) will be accepted by $Y$ and used in encoding new blocks.

When $\mathbf{e}$ is discovered as a corrupted block, $Y$ sends a report $ID_X|\mathbf{e}|Seq(\mathbf{e})|\Phi(\mathbf{e})|ID_Y$ to the M-server. To resist malicious modification on the transmitted report, $Y$ appends to the report a HMAC computed with $scrt_Y$ to provide authentication. The M-server verifies if $\mathbf{e}$ is sent by $X$ by checking "how much $\Phi(\mathbf{e})$ matches $\mathbf{e}, Seq(\mathbf{e})$". In particular, the M-server sets a counter to be zero. For each $\gamma_j \in \overline{\Upsilon}(scrt_X, scrt_Y)$, if $v_j$ is equal to $trunc_\pi(\mathcal{H}(\mathbf{e}|Seq(\mathbf{e}), \gamma_j))$, the counter is incremented. If the counter is finally equal to or larger than a threshold $\theta$, $0 \le \theta \le \lambda - \delta$ (implying that "$\Phi(\mathbf{e})$ very matches $\mathbf{e}, Seq(\mathbf{e})$"), the M-server confirms that $\mathbf{e}$ is sent by $X$. Otherwise ("$\Phi(\mathbf{e})$ does not quite match $\mathbf{e}, Seq(\mathbf{e})$"), $\mathbf{e}$ is judged as a faked block that is not sent by $X$.

The purpose of using the threshold $\theta$ to verify the evidence code is to enable the M-server to detect the potential cheating behaviors of both $X$ and $Y$. To show this, we consider two extreme cases. (1) When $\theta = \lambda - \delta$, the M-server rejects the report as long as a single value (say $v_{j'}$, s.t., $\gamma_{j'} \in \overline{\Upsilon}(scrt_X, scrt_Y)$) in $\Phi(\mathbf{e})$ is not equal to $trunc_\pi(\mathcal{H}(\mathbf{e}|Seq(\mathbf{e}), \gamma_{j'}))$. In this scenario, although $X$ does not know $\Upsilon(scrt_X, scrt_Y)$, it can let $Y$ accept $\mathbf{e}$ with an invalid evidence code $\Phi(\mathbf{e})'$ with high probability, where $\Phi(\mathbf{e})'$ consists of one bogus value and $\lambda - 1$ correct values that $X$ honestly computes using $\mathbf{e}, Seq(\mathbf{e})$ and $k_i$'s, $1 \le i \le \lambda$. Note that $\mathbf{e}$ could be a corrupted block, but the report of $Y$ will be rejected by the M-server. (2) When $\theta = 0$, the M-server does not verify any values in $\Phi(\mathbf{e})$, and consequently a malicious $Y$ can easily disparage $X$ with a bogus block $\mathbf{e}'$ and a randomly generated evidence code $\Phi(\mathbf{e}')$. In the following, we will show that by properly tuning the parameters of $\theta$, $\lambda$, and $\delta$, we can make the probability that a malicious party (either $X$ or $Y$) succeeds be very small.

### A. Security Analysis

When $Y$ is malicious, trying to cheat the M-server with a block $\mathbf{e}'$ not sent by $X$, $Y$ must ensure that there are at least $\theta$ correctly computed values in $\Phi(\mathbf{e}')$, for which it does not have the corresponding knowledge (i.e., $\overline{\Upsilon}(scrt_X, scrt_Y)$) to compute. Since the probability for $Y$ to correctly guess one such value is $(\frac{1}{2})^\pi$. Straightforwardly, we have

**Theorem 1** (Non-repudiation of the recipient) *$Y$ can successfully disparage $X$ with a block not sent by $X$, with probability no larger than* $\sum_{i=\theta}^{\lambda-\delta} \binom{\lambda-\delta}{i} (\frac{1}{2})^{\pi \cdot i} (1 - (\frac{1}{2})^\pi)^{\lambda-\delta-i}$.

As for a malicious $X$ that tries to let $Y$ accept a corrupted block $\mathbf{e}$ with an invalid evidence code $\Phi(\mathbf{e})'$, as we discussed before, $X$ will fail as long as $Y$ detects any inconsistent value in $\Phi(\mathbf{e})'$ with $\Upsilon(scrt_X, scrt_Y)$ or the M-server finds over $\theta$ consistent values with $\overline{\Upsilon}(scrt_X, scrt_Y)$. Therefore, we have

**Theorem 2** (Non-repudiation of the sender) *$X$ can cheat $Y$ with a corrupted block, with probability no larger than*

$\max_{\delta \le x \le \delta+\theta-1} p(x)$, *where* $p(x) = \sum_{i=x-\theta+1}^{\delta} \frac{\binom{\delta}{i}\binom{\lambda-\delta}{x-i}}{2^{\pi\cdot(\delta-i)}\binom{\lambda}{x}}$. ∎

*Proof:* See Appendix 1.

Table II lists the probabilities that a malicious party succeeds in our protocol under several sample parameter selections. We can see that our protocol introduces small space overhead – about 20 bytes per block, which is much smaller than public key signatures (like RSA-1024). Besides, for each incoming block, the receiving node only needs to perform $20 \sim 30$ hash function computations, which incurs extremely short online computational delay. These salient benefits enable us to construct a highly efficient scheme to secure the network coding based P2P streaming network (shown in the next section).

TABLE II: Sample parameter selections

| $\pi$ (bits) | $\lambda$ | $\delta$ | $\theta$ | Prob_X | Prob_Y | Space Ovhd |
|---|---|---|---|---|---|---|
| 8 | 21 | 11 | 4 | $2^{-18}$ | $2^{-23}$ | 22 bytes/blk |
| 6 | 24 | 12 | 5 | $2^{-19}$ | $2^{-20}$ | 19 bytes/blk |
| 6 | 28 | 14 | 6 | $2^{-22}$ | $2^{-24}$ | 22 bytes/blk |

Prob_X (or Prob_Y) – the probability that a malicious $X$ (or $Y$) succeeds. The space overhead includes $\Phi(\mathbf{e})$ and $Seq(\mathbf{e})$ (one byte for $Seq(\mathbf{e})$).

## V. Full Description of the Scheme

Our scheme (MIS) is based on the proposed non-repudiation transmission protocol (Sec. IV). Its basic idea is as follows.

Following the non-repudiation transmission protocol, each peer $X$ registers a secret $scrt_X$ at the M-server, which provides $X$ a list of neighboring peers $N_1, N_2, \cdots, N_T$ together with the partial information of the secrets of these peers, i.e., $\Upsilon(scrt_{N_1}, scrt_X), \cdots, \Upsilon(scrt_{N_T}, scrt_X)$. $X$ uses $scrt_X$ to generate an evidence code for each block it produces, and relies on $\Upsilon(scrt_{N_j}, scrt_X)$ to verify the blocks received from $N_j$.

A node $Y$ decodes the original segment $\mathcal{S}_i$ after receiving $m$ blocks in $\mathcal{S}_i$. If the decoding result does not match the formats of the video streams (used by the media player in playback), $Y$ sends an alert with the segment number $i$ to the M-server with certain probability. The purpose of probabilistically sending alerts is to avoid overwhelming the M-server. An ACK message (containing $i$) is broadcast by the M-server to the overlay (including the S-server) in response to the received alert. If $Y$ does not send an alert and receive no ACK after $\tau$ time units (where $\tau$ is slightly larger than mRTT (the maximum Round-Trip Time between a peer and the M-server)), it keeps increasing the probability of sending an alert until it has sent out an alert or received an ACK.

After receiving the ACK, the S-server generates a checksum for $\mathcal{S}_i$ and disseminates the checksum to the nodes using the streaming overlay (the generation of the checksum will be presented in the scheme description (Sec. V-A)).

Once a node $Y$ receives the checksum, it uses the checksum to verify if any of its received blocks in $\mathcal{S}_i$ is corrupted. A discovered suspicious neighbor $X$ that sent it a corrupted block in $\mathcal{S}_i$ will be reported to the M-server, and then $Y$ forwards the checksum to $X$. If no corrupted blocks are discovered, $Y$ forwards the checksum to all its downstream neighbors.

As we discussed in Sec. III, a suspicious node that cannot discover another suspicious peer is a malicious node. Finally, the M-server broadcasts the result of identified malicious nodes to the overlay, and each node receiving the result adds the malicious nodes that are its neighbors (if any) into its blacklist and will stop accepting any blocks from them in the future.

### A. Scheme Description

Now we present MIS (some details that are duplicated in Sec. IV are omitted for simplicity). Let $PK_M, PK_S$ (or $SK_M, SK_S$) denote the public keys (or private keys) of the M-server and the S-server, respectively. We assume each node can reliably obtain $PK_M, PK_S$ from a trusted PKI. Let $\mathcal{E}(\cdot, Key)$ denote public-key encryption using $Key$, and $\mathcal{E}_{Key}(\cdot)$ denote symmetric-key encryption with $Key$.

1) **Bootstrapping a new node $X$.** $X$ chooses a random $scrt_X$ that is long enough to resist cryptographic attacks, and sends $ID_X|\mathcal{E}(scrt_X, PK_M)$ to the M-server. The M-server selects $T$ peers $N_1, N_2, \cdots, N_T$ in the overlay ($T$ is a system configuration parameter), and sends $\mathcal{E}_{scrt_X}(\{ID_{N_j}, \Upsilon(scrt_{N_j}, scrt_X) : j \in [1,T]\})$ back to $X$. Then $X$ decrypts the bootstrapping information with $scrt_X$.

2) **Sending blocks.** Suppose $X$ has received $t$ blocks $\mathbf{e}_{1,i}, \cdots, \mathbf{e}_{t,i}$ in $\mathcal{S}_i$ and needs to produce a block $\mathbf{e}_{t+1,i}$ for peer $Y$. $X$ first computes $Seq(\mathbf{e}_{t+1,i}) = \max_{1\le k\le t}\{Seq(\mathbf{e}_{k,i})\}+1$, then generates $\Phi(\mathbf{e}_{t+1,i})$ based on $\mathbf{e}_{t+1,i}$ and $Seq(\mathbf{e}_{t+1,i})$, and finally transmits $\mathbf{e}_{t+1,i}|Seq(\mathbf{e}_{t+1,i})|\Phi(\mathbf{e}_{t+1,i})$ to $Y$.

3) **Receiving blocks.** When $\mathbf{e}_{t+1,i}|Seq(\mathbf{e}_{t+1,i})|\Phi(\mathbf{e}_{t+1,i})$ is received by $Y$, $Y$ uses $\Upsilon(scrt_X, scrt_Y)$ to verify if $\Phi(\mathbf{e}_{t+1,i})$ is valid to $\mathbf{e}_{t+1,i}$ and $Seq(\mathbf{e}_{t+1,i})$. If not, $\mathbf{e}_{t+1,i}$ is dropped directly. Otherwise, $ID_X|\mathbf{e}_{t+1,i}|Seq(\mathbf{e}_{t+1,i})|\Phi(\mathbf{e}_{t+1,i})$ is cached in the receiving buffer.

4) **Detecting corrupted blocks in decoding.** Decoding operation is performed by node $Y$ as soon as it has received $m$ blocks in $\mathcal{S}_i$. The decoding result is correct (i.e., identical with the original segment) only when there is no corrupted block in these $m$ blocks. We let each node detect corrupted blocks by checking if the decoding result matches the formats of the video stream, which are used by the media player in playback.

This detection may have false negative cases (although the probability is very small), in which corrupted blocks exist but the decoding result has matching formats. Note that, our goal is just to let the M-server be aware of the existence of malicious nodes. Since any decoding node that detects corrupted blocks will send an alert to the M-server, this approach can detect the existence of malicious nodes with very high probability, due to the fact that it is infeasible for the malicious nodes to design and supply corrupted blocks to ensure that each node in the overlay obtains a decoding result with matching formats. [2]

5) **Sending alerts.** If $Y$ detects corrupted blocks in decoding $\mathcal{S}_i$, it stops generating any new block in $\mathcal{S}_i$ for its downstream

[2]Concerning the false negative cases, the received blocks cannot be immediately freed from the receiving buffer after the decoding, since they will help the nodes (that have received corrupted blocks) identify suspicious neighbors. As to be shown later, it takes at most several mRTTs for the M-server to broadcast an ACK message in response to some nodes' alerts. If no ACK is received, then it is safe to erase these blocks from the receiving buffer; otherwise, they are cached until being verified by the checksum or the M-server disseminates the result of identifying malicious nodes.

neighbors. Besides, $Y$ sends an alert $i|ID_Y|AuthInfo\_Y$ to the M-server with probability $\beta$, where $AuthInfo\_Y$ is a HMAC computed with $scrt_Y$ over the segment number $i$ and $ID_Y$. If $Y$ does not send an alert, it starts a timer and waits for the ACK message (that is broadcast by the M-server in response to some others' alerts). If $Y$ does not receive an ACK after $\tau$ time units, it doubles the probability $\beta$ of sending an alert. This process is repeated until $Y$ has sent out an alert or received an ACK. It is not hard to see that the maximum waiting time for $Y$ to send out an alert is $(1 + \lceil \log_2 \frac{1}{\beta} \rceil) \cdot \tau$.

6) **Broadcasting ACK.** When the M-server receives an alert $i|ID_Y|AuthInfo\_Y$ from $Y$, it first validates $AuthInfo\_Y$ using $scrt_Y$. If this is the first received alert for $\mathcal{S}_i$, the M-server broadcasts an ACK message (containing the sequence number $i$) signed with $SK_M$ to the overlay (including the S-server). Any later received alerts for $\mathcal{S}_i$ are ignored.

7) **Generating checksum.** After the S-server receives and verifies the signed ACK message, it generates a checksum for $\mathcal{S}_i$. We adopt the approach in [9] to construct the checksum. Recall that $\mathcal{S}_i$ consists of $m$ blocks $\mathbf{b}_{1,i}, \cdots, \mathbf{b}_{m,i}$, where $\mathbf{b}_{j,i} = (b_{1,j,i}, \cdots, b_{d,j,i})^\top$, $1 \le j \le m$. The S-server first picks a random seed $\Gamma$, and applies hash function $\mathcal{H}(\cdot)$ on $\Gamma$ to generate $d$ random numbers $s_k = \mathcal{H}(\Gamma, k)$, $1 \le k \le d$. Then, for each block $\mathbf{b}_{j,i}$, $1 \le j \le m$, the S-server computes $u_j = \sum_{k=1}^{d} s_k b_{k,j,i}$, and $(\Gamma, u_1, \cdots, u_m)$ forms the checksum.

The checksum can verify any coded block in $\mathcal{S}_i$. We suppose the block to be verified is $\mathbf{e}_{l,i} = (e_{1,l,i}, \cdots, e_{d,l,i})$, whose coefficient vector is $(c_{1,l,i}, \cdots, c_{m,l,i})$. Then the verification on $\mathbf{e}_{l,i}$ is to check if the value of $\sum_{k=1}^{d} s_k e_{k,l,i}$ is equal to $\sum_{j=1}^{m} c_{j,l,i} u_j$, where the $u_j$'s are reconstructed from $\Gamma$. The correctness of this verification is based on $\sum_{k=1}^{d} s_k e_{k,l,i} = \sum_{k=1}^{d} s_k (\sum_{j=1}^{m} c_{j,l,i} b_{k,j,i}) = \sum_{j=1}^{m} c_{j,l,i} (\sum_{k=1}^{d} s_k b_{k,j,i}) = \sum_{j=1}^{m} c_{j,l,i} u_j$ (recall that $\mathbf{e}_{l,i} = \sum_{j=1}^{m} c_{j,l,i} \mathbf{b}_{j,i}$ and $e_{k,l,i} = \sum_{j=1}^{m} c_{j,l,i} b_{k,j,i}$). The verification is performed in GF($n$).

The probability that a corrupted block is verified by the checksum is $\frac{1}{n}$. To decrease this probability, multiple checksums for $\mathcal{S}_i$ can be used simultaneously. With $\mu$ checksums, the probability that a corrupted block can pass the verification is only $\frac{1}{n^\mu}$ (e.g., $2^{-24}$ when $n = 256$ and $\mu = 3$).

8) **Identifying malicious nodes.** The generated checksum is signed by the S-server and disseminated to the M-server and the nodes using the streaming overlay. When a node $Y$ receives the checksum, it first verifies the appended signature, and then uses the checksum to verify the received blocks in $\mathcal{S}_i$. If all these blocks are consistent with the checksum, $Y$ frees them from the receiving buffer and forwards the checksum to its downstream neighbors. If a corrupted block $\mathbf{e}_{l,i}|Seq(\mathbf{e}_{l,i})|\Phi(\mathbf{e}_{l,i})$ sent by peer $X$ is discovered, $Y$ sends a report $ID_X|\mathbf{e}_{l,i}|Seq(\mathbf{e}_{l,i})|\Phi(\mathbf{e}_{l,i})|ID_Y|AuthInfo\_Y$ to the M-server, and then forwards the checksum to $X$.

Upon receiving the report, the M-server first verifies $AuthInfo\_Y$, and checks if $\mathbf{e}_{l,i}$ is a corrupted block using the checksum received from the S-server. Then the M-server judges whether $\Phi(\mathbf{e}_{l,i})$ is valid to $\mathbf{e}_{l,i}$ and $Seq(\mathbf{e}_{l,i})$, following the non-repudiation transmission protocol. If not, the report is ignored. Otherwise, $X$ is added into the *suspicious node list*. If $X$ also reports a suspicious peer with a valid evidence code, the M-server then removes $X$ from the suspicious node list.

Since $Y$ sends the checksum to $X$ right after reporting $X$, $X$ should be able to report a suspicious node to the M-server no later than mRTT time units after $X$ is added to the suspicious node list. Therefore, the M-server can finish the process of identifying malicious nodes after the suspicious node list is unchanged for mRTT time units.

9) **Releasing the result.** The M-server signs and broadcasts the result of identified malicious nodes to the overlay. After receiving the result, each node $Y$ erases the received blocks in $\mathcal{S}_i$ from the receiving buffer and adds the malicious nodes that are its upstream neighbors (if any) into its blacklist. In the following streaming transmissions, $Y$ will not accept any blocks from these malicious neighbors.

### *B. Discussions*

**Collusion attacks.** We consider multiple malicious nodes collude to hide themselves or disparage innocent nodes. If a colluding malicious node $Y$ is a downstream neighbor of another malicious node $X$, then $Y$ can choose not to report $X$. However, $X$ will still be identified if it sends bogus blocks to any innocent peer. If $X$ sends bogus blocks only to $Y$ and $Y$ further produces corrupted blocks to its downstream neighbors, then $Y$ will be discovered and the pollution flow from $X$ to $Y$ will be stopped at $Y$, without influencing any innocent nodes. If $X$, $Y$ are not directly conneted, they cannot help each other hide themselves, since the reporting and tracking processes are performed by their downstream innocent peers without involvment of $X$, $Y$.

In addition, colluding malicious nodes may try to prevent their downstream peers from receiving the disseminated checksum. However, this strategy is applicable only when there are enough colluding malicious nodes that can entirely isolate these peers from the overlay. One way to address this is to let each node contact the M-server directly to update its neighboring peers after suffering from the incapability of obtaining the checksum or desired video content for a certain period of time.

Besides, malicious nodes may collude to disparage innocent nodes (say $Z$). However, even by collusion, they cannot obtain $scrt_Z$ due to non-invertability of the one-way hash function, and thus they are unable to generate a valid evidence code for a corrupted block $\mathbf{e}'$ and report that $\mathbf{e}'$ is sent by $Z$.

**Non-functional malicious nodes.** We say a node is a *non-functional malicious node* if it exhibits malicious behaviors but replacing it with a legitimate node will not change the set of infected nodes. For example, in Fig. 7, $D$ is a non-functional malicious node. Our currently presented scheme MIS only guarantees identifying all functional malicious nodes (e.g., $B$), whose behaviors cause harm to innocent nodes. In fact,

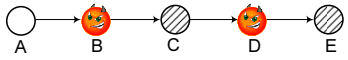


Fig. 7: An exmaple of non-functional malicious nodes ($D$).

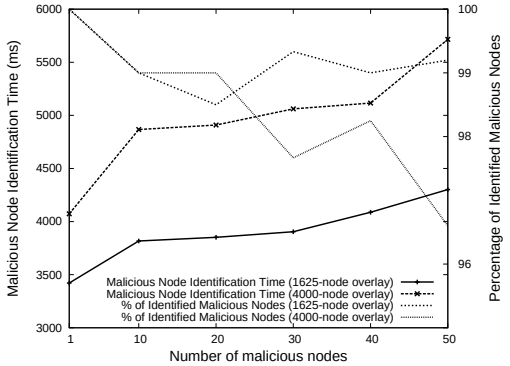


Fig. 8: Malicious node identification times and the pertentage of identified malicious nodes.

we can rely on some advanced strategies (by recomputing the checksum based on the discovered corrupted blocks) to detect non-functional malicious nodes, but due to the space limitation we explain this in detail in our journal paper.

**Sending fake alerts.** Sending fake alerts may trigger the system to meaninglessly search for malicious nodes, wasting system resources. For this, we let the M-server punish those fake alert senders by treating them as malicious nodes if no malicious node are discovered at the end of the search.

**Transmission errors.** MIS can tolerate transmission errors and even malicious modifications on the transmitted blocks, because each receiving node verifies the evidence code appended with the received block and a block with errors or modifications will be directly rejected, without influencing the decoding operations or the reporting/identification processes.

**Node churn.** Node churn is common in P2P networks. First, if malicious nodes churn after injecting corrupted blocks, our scheme can still identify them, since the process of identifying malicious nodes is based on the evidence codes generated before and does not require the involvement of malicious nodes. Second, if an innocent node $Y$ – who is a downstream neighbor of a malicious node $X$ – churns, then $Y$ may not report $X$ to the M-server. However, as long as any other downstream peer of $X$ is alive, $X$ will be identified.

## VI. Evaluation

### A. Simulation

We simulate MIS in Java based on real PPLive channel overlays with $1,600 \sim 4,000$ nodes, which are obtained in our previous work [19]. We choose random propagation delay for each overlay link in the range of $[100ms, 500ms]$. The computational delay for each node to process a checksum is $5ms \sim 10ms$. The malicious nodes are selected at random out of all the nodes, and each of them injects bogus blocks to all its downstream neighbors. There is one S-server that serves 5% peers directly. Each segment consists of 32 blocks, and each block has 256 codewords in GF(256) (following the configurations in [2]). Fig. 8 gives the time that MIS takes to identify malicious nodes as well as the percentage of the malicious nodes that are discovered. The results are averaged over 10 independent runs.

We can see that over 96% of malicious nodes are identified by MIS, and those that are not discovered are non-funcational malicious nodes as we discussed before. In addition, our scheme only takes several seconds to identify these malicious nodes, which implies that the system can recover quickly.

TABLE III: Comparisons

| Schemes | Comp. efficiency | Space ovhd | Verif. info. distribution |
|---|---|---|---|
| Homo. hash [8] | Low | Large | Repeated |
| Prob. check [9] | Low | Large | Repeated |
| Trapdoor hash [10] | Low | Large | Once |
| Homo. sig. [12] | Low | Large | Once |
| MAC-based [14] | High | Very large | Once |
| Null key [13] | Very high | Large* | Repeated |
| MIS (ours) | Very high | Very Small | Once |

*Due to the repeatedly distributed verification information. The space overhead rate is $\frac{d}{m}$, where $d$ is the number of codewords contained in a block and $m$ is the number of blocks in a segment.

### B. Comparison

We compare the online performance of MIS against the existing schemes based on the on-the-fly verification [8]–[10], [12]–[14] [3]. Among these previous schemes, the null key [13] has the highest computational efficiency, which takes $1 \sim 2us$ to check a block on our machine with 2.2 Ghz dual CPUs. Yu et al. [14] show that their MAC-based scheme takes about $2ms$ to sign or verify a block. The schemes constructed from homomorphic crypto-systems incur much longer delays, which are over $1s$ according to the results given in [14]. MIS takes about $5us$ to check an incoming block and $10us$ to generate an evidence code for an outgoing block (implemented using Miracl with SHA-256 on the same machine) with the parameters given in Table II. In addition, MIS introduces the smallest space overhead – only 22 bytes per block. Whereas, the space overhead in [8], [9], [12] is 128 bytes/block, and is 256 bytes/block in [10]. The MAC-based scheme [14] incurs much larger space overhead, wihch is $20\% \sim 40\%$ of the original blocks. Furthermore, MIS does not require repeatedly distributing verification information, further reducing the communication overheads and processing costs. Table III summarizes the comparisons.

## VII. Related Work

Krohn et al. [8] initiated the study of network-coding pollution attacks and proposed an on-the-fly verification scheme based on homomorphic hashing, which has high computational expense. To mitigate computational costs, Gkantsidis and Rodriguez [9] proposed to probabilistically check blocks using Krohn et al.'s scheme [8], but their scheme introduces larger communication overheads. Recently, Kehdi and Li [13] proposed a light-weight scheme based on the null-space property of network coding. One drawback of this scheme is the

[3]We do not compare our scheme with the error-correction based schemes since they cannot guarantee the full usage of network capacity and are applicable only when a limited number of blocks are corrupted.

vulnerability to collusion attacks, where multiple malicious nodes can collude to infer the null keys employed in the network and let innocent nodes accept corrupted blocks. In all these schemes [8], [9], [13], the verification information is derived from the blocks and thus needs to repeatedly pre-distributed via secured channels.

To avoid redistributing verification information, Li et al. [10] proposed a scheme based on trapdoor one-way permutation, in which only a random seed needs to be pre-distributed and each block carries a verificable pad computed using the seed and the trapdoor information. Charles et al. [11] and Yu et al. [12] proposed homomorphic signatures for network coding. However, these schemes [10]–[12] require expensive modular exponentiation computations at each hop, which is unallowable for live streaming applications. Yu et al. [14] proposed a computationally efficient scheme for XOR network coding based on symmetric keys. However, this scheme incurs substantial communication overheads.

Some schemes deal with corrupted blocks at decoders. Ho et al. [15] proposed a scheme that can detect Byzantine errors in decoding, but cannot correct them. The schemes in [16] and [17] can correct Byzantine errors, but they are applicable only when less than a threshold number of bogus blocks injected into the network and the achievable flow rate is determined by the number of contaminated links. These schemes [15]–[17] cannot limit the behaviors of malicious nodes, which can persistently degrade the network performance by continuously injecting bogus blocks. Whereas, our scheme can actively identify malicious nodes and remove them from the network.

Identifying malicious nodes have been studied in several areas, such as botnet detection [20], Byzantine link failure localization [21], and reputation-based networks [22]. These approaches typically require monitoring the network traffic or nodes' behaviors for a certain period of time and rely on some statistical algorithms to identify Byzantine adversaries. On the contrary, our scheme is message-based in that the malicious nodes can be rapidly located as long as they inject a single false message into the network.

## VIII. Conclusion

In this paper, we propose a novel scheme (MIS) to limit network-coding pollution attacks by identifying malicious nodes. MIS can fully satisfy the requirements of P2P live streaming systems. It has high computational efficiency, small space overhead, and the capability of handling a large number of corrupted blocks and malicious nodes, and does not require repeatedly pre-distributing verification information. MIS is block-based and can repaidly identify malicious nodes, ensuring that the system quickly recovers from pollution attacks.

## References


[1] X. Zhang, J. Liu, B. Li, and T.-S.P. Yum, "Data-driven Overlay Streaming: Design, Implementation, and Experience", in *IEEE INFOCOM*, 2005.

[2] M. Wang, and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming", in *Proc. of IEEE INFOCOM*, 2007.

[3] M. Wang, and B. Li, "$R^2$: Random Push with Random Network Coding in Live Peer-to-Peer Streaming ", in *IEEE Journal on Selected Areas in Communications*, vol.25(9), pp. 1655-1666, 2007.

[4] C. Feng, and B. Li, "On Large-Scale Peer-to-Peer Streaming Systems with Network Coding", in *ACM Multimedia (MM'09)*, 2009.

[5] R. Ahlswede, N. Cai, S.-Y.R. Li, and R. W. Yeung, "Network Information Flow", in *IEEE Trans. Inf. Theory*, vol. 46(4), pp. 1204-1216, 2000.

[6] R. Koetter, and M. Medard, "An Algebraic Approach to Network Coding", in *IEEE/ACM Trans. on Networking*, vol. 11(5), pp. 782-795, 2003.

[7] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Settings", *ISIT'03*, 2003.

[8] M. Krohn, M. Freeman, and D. Mazieres, "On-the-fly Verification of Rateless Erase Codes for Efficient Content Distribution", in *Proc. IEEE Symp. on Security and Privacy (Oakland)*, 2004.

[9] C. Gkantsidis, and P. R. Rodriguez, "Cooperative Security for Network Coding File Distribution", in *Proc. of IEEE INFOCOM*, 2005.

[10] Q. Li, D.-M. Chiu, and J. C. S. Lui, "On the Practical and Security Issues of Batch Content Distribution Via Network Coding", in *Proc. of IEEE International Conference on Network Protocols (ICNP'06)*, 2006.

[11] D. Charles, K. Jain, and K. Lauter, "Signatures for Network Coding", in *40th Annual Conf. on Infomation Science and Systems (CISS'06)*, 2006.

[12] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An Efficient Signature-based Scheme for Securing Network Coding against Pollution Attacks", in *Proc. IEEE INFOCOM*, 2008.

[13] E. Kehdi, and B. Li, "Null Keys: Limiting Malicious Attacks Via Null Space Properties of Network Coding", in *Proc. of IEEE INFOCOM*, 2009.

[14] Z. Yu, Y. Wei, B. Ramkumar, Y. Guan, "An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks", *IEEE INFOCOM*, 2009.

[15] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. R. Karger, "Byzantine Modification Detection in Multicast Networks using Randomized Network Coding", *ISIT'04*, 2004.

[16] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient Network Coding in the Presense of Byzantine Adversaries", in *Proc. of IEEE INFOCOM*, 2007.

[17] R. Kotter, and F. R. Kschischang, "Coding for Errors and Erasures in Random Network Coding", *ISIT'07*, 2007.

[18] M. Wang, and B. Li, "How Practical is Network Coding?", in *Proc. of IEEE International Workshop on Quality of Service (IWQOS'06)*, 2006.

[19] L. Vu, I. Gupta, K. Nahrstedt, and J. Liang, "Understanding the Overlay Characteristics of a Large-scale Peer-to-Peer IPTV System", to appear in *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 2009.

[20] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic", *NDSS'08*, 2008.

[21] B. Awerbuch, D. Holmer, C. N-. Rotaru, and H. Rubens, "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures", in *Proc. of the 1st ACM workshop on Wireless security (WiSe'02)*, 2002.

[22] E. Damiani, D.C. Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks", *CCS'02*, 2002.

[23] R. Wang, W. Du, and P. Ning, "Containing Denial-of-Service Attacks in Broadczast Authenticatin in Sensor Networks", *MobiHoc'07*, 2007.

[24] Y. Huang, W. He, K. Nahrstedt, and W. C. Lee, "DoS-Resistant Broadcast Authentication Protocol with Low End-to-end Delay", in *2nd IEEE Workshop on Mission-Critical Networking*, 2008.


**Appendix 1** (Proof of Theorem 2)

*Proof:* Let $x$ denote the number of correctly computed values in $\Phi(\mathbf{e})'$. On the one hand, $x$ should be smaller than $\delta+\theta$; otherwise, there will be at least $\theta$ values in $\Phi(\mathbf{e})'$ that pass the M-server's checking. On the other hand, $x$ should be at least $\delta$. Otherwise, at least $x-\delta$ inconsistent values will be detected by $Y$. Hence, $x \in [\delta, \delta+\theta-1]$. Let $i$ be the number of correctly computed values (in $\Phi(\mathbf{e})'$) that are verificable to $Y$. Hence, the number of consistent values that are checked by the M-server is $x-i$, which should be smaller than $\theta$, i.e., $i \geq x-\theta-1$. Therefore, the probability that $Y$ accepts the corrupted block when there are $x$ ($\in [\delta, \delta+\theta-1]$) consistent values in $\Phi(\mathbf{e})'$ is $p(x) = \sum_{i=x-\theta+1}^{\delta} \frac{\binom{\delta}{i}\binom{\lambda-\delta}{x-i}}{2^{\pi(\delta-i)}\binom{\lambda}{x}}$, and the best probability is $\max_{\delta \leq x \leq \delta+\theta-1} p(x)$. ■