MIS: Malicious Nodes Identification Scheme in Network-Coding-Based Peer-to-Peer Streaming

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

Abstract—Network coding has been shown to be capable of greatly improving 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 are combined with other legitimate blocks at downstream nodes, leading to incapability of decoding the original blocks and substantial degradation of network performance. In this paper, we propose a novel approach to limiting pollution attacks by rapidly identifying malicious nodes. 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. 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

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 [3], [4] found that network coding can greatly improve 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, as an alternative paradigm of traditional “store-and-forward” routing, allows participating nodes to code incoming data blocks (typically by linear combination). However, the “combination” nature of network coding makes it vulnerable to pollution attacks, where malicious nodes inject into the network bogus blocks that are combined with other legitimate blocks at downstream nodes, leading to incapability of decoding the original blocks and substantial degradation of network performance. Thus, network coding cannot be safely applied to P2P streaming networks, unless the problem of pollution attacks is addressed.

Several schemes dealing with network-coding pollution attacks have been proposed in the literature, but none of them is applicable for real-time streaming. Homomorphic signature or hashing schemes [5], [7]–[9] enable intermediate nodes to verify blocks on the fly, but introduce considerable computational delays. Although symmetric key based scheme [11] and the scheme based on null space properties of network coding [10] achieve much higher computational efficiency, they either introduce large communication overheads or require repeatedly pre-distributing verification information to all the nodes in the overlay. Error correction based approaches [13], [14] provide error-tolerant decoding at sink nodes. 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.

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 rapidly 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.

MIS can fully satisfy the requirements of live streaming. In MIS each node only needs to perform a small number of hash computations for an incoming/outgoing block, incurring 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, MIS is scalable to large networks and is effective even in the presence of a large number of malicious nodes. We simulate MIS based on real PPLive channel overlays with 1,600 ~ 4,000 nodes, which are obtained in our previous work [16]. The simulation results show that the process of identifying malicious nodes in MIS only takes a few seconds.

This paper is organized as follows: Section II introduces the background and models. Section III describes our scheme MIS. Section IV presents the non-repudiation transmission protocol. Section V gives the evaluation results and compares MIS with existing schemes. Section VII concludes the paper.

II. BACKGROUND AND MODELS

A. Background of P2P Streaming with Network Coding

We consider a mesh-structured P2P streaming network that involves a large number of overlay nodes and a streaming server
that distributes live videos to the overlay. Each live stream is divided into segments $S_i$, $i = 1, 2, \cdots$, 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 $S_i$ is further divided into $m$ blocks $b_{1,i}, \cdots, b_{m,i}$, and each block $b_{j,i}$ is subdivided into $d$ codewords $b_{j,i} = (b_{j,1,i}, \cdots, b_{j,d,i})^T$, $1 \leq j \leq m$. The segment $S_i$ is considered as an $d \times m$ matrix of elements of the Galois field $GF(n)$ (e.g., $n = 256$), as shown below.

$$S_i = (b_{1,i}, \cdots, b_{m,i}) = \begin{pmatrix} b_{1,1,i} & \cdots & b_{1,m,i} \\ \vdots & \ddots & \vdots \\ b_{d,1,i} & \cdots & b_{d,m,i} \end{pmatrix}$$

When the server attempts to send a block $e_{1,i}$ in $S_i$ 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 server creates the coded block $e_{1,i}$ by linearly combining the original blocks $b_{1,i}, \cdots, b_{m,i}$ with $(c_{1,1,i}, \cdots, c_{m,1,i})$, i.e., $e_{1,i} = \sum_{j=1}^{m} c_{j,1,i} b_{j,i} = (\sum_{j=1}^{m} c_{j,1,i} b_{j,1,i}, \cdots, \sum_{j=1}^{m} c_{j,1,i} b_{j,d,i})$. In other words, $e_{1,i}$ is obtained by multiplying its coefficient vector $(c_{1,1,i}, \cdots, c_{m,1,i})$ with the $d \times m$ matrix $S_i$ in $GF(n)$. The coefficient vector $(c_{1,1,i}, \cdots, c_{m,1,i})$ is appended to the coded block $e_{1,i}$ and the augmented block is sent to peer $X$.

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

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 that the server is trusted, and publicly known so that each node can contact it directly. We assume there exists a reliable PKI that enables each node to securely obtain the server’s public key. The public key is used by the server to broadcast authenticated information (e.g., the result of identified malicious nodes) to the overlay. Broadcast authentication is an important security primitive and has been extensively studied, but it is orthogonal to this work.

III. MIS: MALICIOUS NODE IDENTIFICATION SCHEME FOR NETWORK CODING

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. Our goal is to track the origin of corrupted blocks.

A. Scheme Description

The first step of our scheme is to detect the existence of malicious nodes. We let each decoding node detect 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 server to trigger the process of identifying malicious nodes. The alert contains the sequence number of the polluted segment, a time stamp, and the reporting node’s ID. To prevent malicious injection or modification on transmitted alerts, each alert is appended with a HMAC computed with a secret key shared between the reporting node and the server (assuming that each node registers a secret key at the server when it joins the overlay). Since it is extremely hard for malicious nodes to design and supply corrupted blocks to ensure that each node in the overlay obtains a decoding result with matching formats, from the standpoint of the whole overlay, the existence of malicious nodes can be detected with sufficiently high probability.

After receiving an alert, the server computes a checksum based on the original blocks in the polluted segment. With this checksum, infected nodes can identify which neighbor has sent it a corrupted block. We adopt the approach presented in [6] to construct the checksum, and ensure that any corrupted block can be detected with probability close to 1 (we refer readers to our technical report [1] for details of checksum construction).
The checksum is signed by the server and disseminated to the overlay. For example, in Fig. 1, 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 could be innocent and receive corrupted blocks from their upstream neighbors. For instance, \( J \) is innocent and the corrupted block \( J \) produces due to the bogus block it has received from \( F \); whereas, \( F \) 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 server (This reporting message is also protected by HMAC). Then, the reporting nodes \( (I, K) \) further forward the checksum to their suspected upstream neighbors \( (I, K) \).

If a suspected node is truly 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 server. On the contrary, if a suspected node is malicious \( (e.g., F) \), it cannot find a suspicious neighbor that has sent it a corrupted block. Therefore, we let the server 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 \( ( \text{when being suspected}) \). 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 sign the block it sends out using a public-key signature scheme, and the signature associated with the block can be used as the evidence to demonstrate that the reported node has sent this block. However, this approach requires applying public-key signature on each transmitted block, introducing substantial computational delays due to the expensive signature generation and verification. Alternatively, we design a light-weight non-repudiation transmission protocol \( ( \text{described in Section IV}) \) 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.

B. Discussions

Transmission cycles. In mesh-structured P2P networks, cycles possible exist. Consider a cycle with three nodes \( X \rightarrow Y \rightarrow Z \rightarrow X \), where \( X \) is malicious and \( Y,Z \) are infected by \( X \). In this case, since \( X \) receives a corrupted block from \( Z \) (although the corruption of this block is caused by \( X \), \( X \) can report \( Z \) as a suspicious node to the server, and the server cannot tell which node in the cycle is the origin of corrupted blocks. To address this problem, we let each node append to its transmitted block \( (\text{say } e_{t+1,i}) \) with a sequence number \( ( \text{denoted by } Seq(e_{t+1,i}) \) ), which is set as the maximum sequence number of all of its received blocks \( (e_{1,i}, \cdots, e_{t,i}) \) in this segment plus one, i.e., \( Seq(e_{t+1,i}) = \max_{1 \leq k \leq t} \{Seq(e_{k,i})\} + 1 \), and the evidence associated with \( e_{t+1,i} \) is computed based on \( Seq(e_{t+1,i}) \) and the content of \( e_{t+1,i} \). Then the node that has sent a block with the smallest sequence number among all nodes in the cycle is judged as the malicious node.

Collusion attacks. We first consider multiple malicious nodes collude to hide themselves. If a colluding 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 connected, they cannot help each other hide themselves, since the reporting and tracking processes are performed by their downstream innocent peers without involving \( X \) and \( Y \).

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 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. As to be shown in Section IV, our non-repudiation transmission protocol lets each node compute the evidence of its transmitted block independently, and ensures that any colluding nodes who combine their knowledge cannot gain any advantage of forging any evidence to disparage innocent nodes.

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, consider a transmission path \( X \rightarrow Y \rightarrow Z \rightarrow W \) with two malicious nodes \( X,Z \) and two infected nodes \( Y,W \); then \( Z \) is a non-functional malicious node. Our scheme MIS only guarantees identifying all functional malicious nodes \( (e.g., X) \), whose behaviors cause harm to innocent nodes.

Node churn. Node churn is common in P2P networks. MIS can effectively identify malicious nodes even if they churn right after injecting corrupted blocks, because the process of identifying malicious nodes does not require the involvement of malicious nodes. With a blacklist maintained by the server, MIS could prevent malicious nodes from rejoining the overlay to launch pollution attack again. Besides, if an innocent node \( Y \) – who is a downstream neighbor of a malicious node \( X \) – churns, then \( Y \) may not report \( X \) to the server. However, as long as any other downstream peer of \( X \) is alive, \( X \) will be identified.

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

We let $X$ be the suspicious node, and $Y$ be the reporting node. Let $e$ denote the block that $X$ transmits to $Y$, and $\Phi(e)$ denote the evidence (referred to as evidence code) associated with $e$ and $\text{Seq}(e)$. Then $\Phi(e)$ should convince the server that $e$ is indeed sent by $X$, but not forged by $Y$.

Protocol description. We let each of $X$ and $Y$ initially register a secret (denoted by $\text{scrt}_X$, $\text{scrt}_Y$) with the server, respectively. $X$ will use $\text{scrt}_X$ to produce $\Phi(e)$. The server provides partial information of $\text{scrt}_X$ (denoted as $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$) to $Y$ according to the information of $\text{scrt}_Y$. $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$ can help $Y$ verify $\Phi(e)$.

The generation of $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$ is as follows. First, the server maps $\text{scrt}_Y$ to a $\delta$-element subset $F(\text{scrt}_Y, \text{ID}_X)$ of $\{1, \ldots, \lambda\}$, where $F$ is a pseudo-random mapping function. For example, when $\lambda = 6$, $\delta = 3$, $F(\text{scrt}_Y, \text{ID}_X)$ could be $\{2, 4, 5\}$. Second, the server derives $\lambda$ secret elements from $\text{scrt}_X$ by computing $\gamma_i = H(\text{scrt}_X, \text{ID}_X, i)$, $1 \leq i \leq \lambda$, where $H$ is a secure hash function. Third, the server initializes $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$ as an empty set, and then for each $i^* \in F(\text{scrt}_Y, \text{ID}_X)$ the server adds $\gamma_i$ into $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$. The finally obtained $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$ is given to $Y$. Note that from $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$, $Y$ cannot learn any information about $\text{scrt}_X$ or the secret elements that are derived from $\text{scrt}_X$ but not in $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$ (i.e., $\{\gamma_j : j \in \{1, \ldots, \lambda\}/F(\text{scrt}_Y, \text{ID}_X)\}$, denoted by $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$).

The evidence code $\Phi(e)$ consists of $\lambda$ values $\{v_1, \ldots, v_\lambda\}$, each of which is computed by $v_i = \text{trunc}_{\pi}(H(\text{e}, \text{Seq}(e), \gamma_i))$, $1 \leq i \leq \lambda$, where $\text{trunc}_{\pi}$ is a function that truncates the input into leftmost $\pi$ bits. Once receiving $\text{e|Seq}(e)|\Phi(e)$ from $X$, $Y$ verifies the validity of $\Phi(e)$ by checking if $v_i$ is equal to $\text{trunc}_{\pi}(H(\text{e}, \text{Seq}(e), \gamma_i))$ for all $\gamma_i$’s in $\overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$. Only a block with a valid evidence code will be accepted by $Y$ and used in encoding new blocks.

When $Y$ reports $X$ to the server with $\text{e|Seq}(e)$ and $\Phi(e)$, the server verifies if $e$ is sent by $X$ by checking “how much $\Phi(e)$ matches $e$, $\text{Seq}(e)$”. In particular, the server sets a counter to be zero. For each $\gamma_j \in \overline{\Upsilon}(\text{scrt}_X, \text{scrt}_Y)$, if $v_j$ is equal to $\text{trunc}_{\pi}(H(\text{e}, \text{Seq}(e), \gamma_j))$, the counter is incremented. If the counter is finally equal to or larger than a threshold $\theta$, $0 \leq \theta \leq \lambda - \delta$ (implying that “$\Phi(e)$ very matches $e$, $\text{Seq}(e)$”), the server confirms that $e$ is sent by $X$. Otherwise (“$\Phi(e)$ does not quite match $e$, $\text{Seq}(e)$”), $e$ is judged as a faked block that is not sent by $X$.

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

Theorem 1 (Non-repudiation of the recipient) $Y$ can successfully disapprove $X$ with a block not sent by $X$, with probability no larger than $\sum_{\theta=0}^{\lambda-\delta} \binom{\lambda-\delta}{\theta} \left(\frac{1}{2}\right)^\theta (1 - \frac{1}{2})^{\lambda-\delta-\theta}$.

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

Theorem 2 (Non-repudiation of the sender) $X$ can cheat $Y$ with a corrupted block, with probability no larger than $\max_{\delta \leq x \leq \lambda + \theta - 1} p(x)$, where $p(x) = \sum_{i=x-\theta+1}^{\delta} \binom{\lambda - \delta}{i} \left(\frac{1}{2}\right)^i (1 - \frac{1}{2})^{\lambda - \delta - i}$.


Table I lists the probabilities that a malicious party succeeds in our protocol under several sample parameter selections.

<table>
<thead>
<tr>
<th>Parameter Selections</th>
<th>$\pi$ (bits)</th>
<th>$\lambda$</th>
<th>$\theta$</th>
<th>$\text{Prob}_X$</th>
<th>$\text{Prob}_Y$</th>
<th>Space Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>21</td>
<td>1</td>
<td>2</td>
<td>22</td>
<td>22 bytes/blk</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>24</td>
<td>2</td>
<td>2</td>
<td>22</td>
<td>19 bytes/blk</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>28</td>
<td>4</td>
<td>5</td>
<td>22</td>
<td>22 bytes/blk</td>
</tr>
</tbody>
</table>

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

V. EVALUATION

We simulate MIS in Java based on real PPlive channel overlays with 1,600 ~ 4,000 nodes, which are obtained in our previous work [16]. 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 ~ 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. The server serves 5% peers directly. Each segment consists of 32 blocks, and each block has 256 codewords in GF(256) (following the configurations in [3]).

Fig. 2: Identification times and the percentage of identified malicious nodes. Results are averaged over 10 independent runs.

We can see from Fig. 2 that over 96% of malicious nodes are identified by MIS, and those that are not discovered are non-functional malicious nodes as we discussed before. In addition, the identification process only takes several seconds, which ensures that the system can recover quickly.

We compare the online performance of MIS against the existing schemes based on the on-the-fly verification [5]–[7], [9]–[11]. Among these previous schemes, the null key [10]
TABLE II: Comparisons

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Comp. efficiency</th>
<th>Space ovhd</th>
<th>Verif. info. distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homo. hash [5]</td>
<td>Low</td>
<td>Large</td>
<td>Repeated</td>
</tr>
<tr>
<td>Prob. check [6]</td>
<td>Low</td>
<td>Large</td>
<td>Repeated</td>
</tr>
<tr>
<td>Trapdoor hash [7]</td>
<td>Low</td>
<td>Large</td>
<td>Once</td>
</tr>
<tr>
<td>Homo. sig. [9]</td>
<td>Low</td>
<td>Large</td>
<td>Once</td>
</tr>
<tr>
<td>Null key [10]</td>
<td>Very high</td>
<td>Large*</td>
<td>Repeated</td>
</tr>
<tr>
<td>MIS (ours)</td>
<td>Very high</td>
<td>Very Small</td>
<td>Once</td>
</tr>
</tbody>
</table>

*Due to repeatedly distributing verification information. The space overhead rate is $\frac{d}{m}$.

has the highest computational efficiency, which takes $1 \sim 2\mu s$ to check a block on our machine with 2.2 Ghz dual CPUs. Yu et al. [11] show that their MAC-based scheme takes about $2\mu s$ 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 [11]. MIS takes about $5\mu s$ to check an incoming block and $10\mu s$ 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 I. In addition, MIS introduces the smallest space overhead – only 22 bytes per block. Whereas, the space overhead in [5], [6], [9] is 128 bytes/block, and is 256 bytes/block in [7]. The MAC-based scheme [11] incurs even larger space overhead. Furthermore, MIS does not require repeatedly distributing verification information. Table II summarizes the comparisons.

VI. RELATED WORK

Krohn et al. [5] 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 [6] proposed to probabilistically check blocks using Krohn et al.’s scheme [5], but their scheme introduces larger communication overheads. Recently, Kehdi and Li [10] proposed a light-weight scheme based on the null-space property of network coding. One drawback of this scheme is the 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 [5], [6], [10], 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. [7] proposed a homomorphic hashing scheme based on trapdoor one-way permutation. Charles et al. [8] and Yu et al. [9] proposed homomorphic signatures for network coding. However, these schemes [7]–[9] require expensive modular exponentiation computations at each hop, which is unallowable for live streaming. Yu et al. [11] proposed an 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. [12] proposed a scheme that can detect Byzantine errors in decoding, but cannot correct them. The schemes in [13] and [14] 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.

VII. 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 reliably identify malicious nodes, ensuring that the system quickly recovers from pollution attacks.

VIII. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0524695. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of those agencies.

REFERENCES