A Detection Mechanism for Internal Attacks on Pull-based P2P Streaming Systems

Hatem Ismail
DEEDS Group, TU Darmstadt
hayman@deeds.informatik.tu-darmstadt.de

Stefanie Roos
University of Waterloo
stefanie.roos@uwaterloo.ca

Neeraj Suri
DEEDS Group, TU Darmstadt
suri@cs.tu-darmstadt.de

Abstract—**Online streaming is a popular service for data-intensive applications such as video streaming. P2P-based streaming solutions are advocated to help reduce costs for both providers and users. Yet, involving users over data dissemination entails security risks including a variety of denial-of-service attacks. While extensive research exists on mitigating varied attack types, their effectiveness is limited if the attacker can infer information about the topology such as the identity of nodes that have direct connections to the source. The attacker can then leverage the gained insights to place malicious participants in prominent positions. By dropping chunks that should be forwarded, the malicious peers degrade the performance in a stealthy way that does not raise suspicion.**

We first demonstrate the feasibility of conducting such attacks. Accordingly, we propose a detection mechanism that identifies the attack and removes potential malicious peers from their disruptive positions. We ascertain, theoretically and through simulations, that malicious peers cannot misuse the detection mechanism to gain influence. Our simulation-based study indicates that the proposed detection mechanism is able to detect malicious peers with up to 80-90% accuracy while inducing a small overhead of approximately 8%.

I. Introduction

Streaming content is an essential component of data-driven infrastructures [1]. Most streaming applications rely on (monopolistic) central providers that have significant computing and communication resources to distribute high quality of service content to a large audience in a fast and reliable manner. For the providers, this entails the use of dedicated resources, involves performance and scalability issues along with considerations of handling single point of failures. Distributing content via a Peer-to-Peer (P2P) network significantly lowers the load that the provider experiences as the participants relay the downloaded content to other participants. This eliminates the need that all participants receive their content directly from the source, i.e., the provider. Consequentially, P2P streaming becomes an attractive option for alternative distributed providers and user-generated content to help reduce data delivery costs and results in lower rates for customers.

However, achieving high reliability in a P2P overlay and across a dynamic and heterogeneous group of content distributors is challenging. In addition to the inherent operational unreliability of benign participants, attackers such as competitors can infiltrate the set of peers and conduct denial-of-service attacks. In this manner, malicious participants can interrupt or delay the distribution of the content with the goal of degrading the quality of service.

In general, nodes in a P2P streaming system connect to a small set of other nodes, called the *neighbor set*. The publisher or the *source* of the content divides the stream into *chunks*, which each contain an equal-sized part of the encoded data, and forwards these chunks to its neighbors. Nodes in the system receive chunks from neighbors and forward them to neighbors that have not previously received the respective chunks. The selection of neighbors and the choice of neighbors to receive-from or forward-to differs across protocols [2]. Yet, pull-based mesh networks are the predominant method in P2P streaming systems [3]. In a mesh overlay, peers maintain a buffer-map indicating which chunks they possess. Neighbors periodically exchange their buffer-maps and request chunks from neighbors whose buffer-maps indicate possession of the respective chunk. Peers then forward chunks based on the received requests.

In pull-based systems, there exist several denial-of-service attacks, known as buffer-map or BM cheating attacks [4]. In such an attack, malicious nodes might drop or delay chunks. Alternatively, they might advertise chunks that they do not have. As detection of the latter is locally possible and the effect of delaying chunks is at most as severe as entirely dropping the respective chunks, we focus on a denial-of-service attack through dropping chunks without advertising them, called *Drop-chunk* in the following.

In the past, multiple countermeasures aimed to reduce the severity of the *Drop-chunk* attack. However, the majority of these defences [5]–[7] assume that the attacker is unaware of the topology of the streaming network and specifically does not know the *headnodes*, i.e., the peers connected to the source. However, previous work indicates that it is relatively easy to infer the identity of benign headnodes and then target those important nodes [8]. While countermeasures to these inference attacks exist, they assume an external adversary that can shutdown or replace certain nodes [8]–[10]. Hence, the existing work evaluates neither the impact of nor protection mechanism against internal colluding attackers, i.e., attackers that insert nodes under their control into the system pretending to be regular participants.

In this paper, we first illustrate the effectiveness of internal attacks based on malicious headnodes. Consequently, we propose a mechanism for detecting *Drop-chunk* by keeping track of peer satisfaction. If the cumulative satisfaction level of a group of peers drops below a certain threshold, the source replaces all headnodes associated with the group with randomly chosen peers. Hence, our detection and mitigation mechanism efficiently reacts to a detected low quality of service rather than explicitly identifying the misbehavior of one or more specific nodes. Note that we focus on attackers that control a low fraction of nodes, as attackers controlling the majority of nodes can trivially control most of the communication, even without gaining strategic positions such as headnodes.

Using a simulation study, we show that the proposed detection mechanism effectively restores peers satisfaction, even in a scenario where the attacker controls all headnodes. Our mechanism introduces only a small signaling overhead of approximately 8% supporting the claim of being efficient.

A theoretical analysis complements our practical results, focusing on the opportunities to abuse the detection mechanism. Indeed, the detection algorithm prevents malicious nodes from replacing benign headnodes with malicious nodes unless they control a large fraction of the total number of nodes. Furthermore, maintaining malicious headnodes despite generally low satisfaction levels is not possible for the considered attacker.

II. Internal Attack Model

In this section, we introduce the concept of internal attacks in streaming P2P overlays. Our focus is on attacks on availability that aim to intercept data chunks from the source. We start by introducing the attack characteristics such as target, budget, and malicious nodes placement. Subsequently, our main discussion outlines the *Drop-chunk* adversarial behavior.

A. Target, budget and placement

The target of the internal attack is to severely degrade the user's satisfaction by interrupting the stream close to the source, thus preventing dissemination between benign peers. The budget x of the attacker corresponds to the number of nodes controlled by the malicious party. In accordance with the attack goal of maximizing impact, the attacker aims to use its budget to occupy the source's neighbor list. Note that in a real streaming system, the typical size of the source's neighbor list is 20-30 entries [11], [12], which highlights the feasibility of conducting an internal attack using a very small budget.

We assume an attacker to (a) have a budget x, and (b) be capable of assigning malicious peers as headnodes. Potential ways of assigning headnodes include (1) joining the overlay as early as possible in case of a pre-announced time for a streamline, (2) taking down the source's benign headnodes, or (3) abusing peers' replacement mechanisms [8]. Hence, the attacker initially assigns $x_h \leq x$ of its resources as headnodes. As the attacker's main objective is to fully occupy the source's neighbor list, the optimum value of x_h for the attacker, is $x_h = |NeighborList|$. If full exploitation of the source's neighbor list is not feasible when the attack is being initiated, the attacker continuously tries to increase the value of x_h.

The rest of malicious peers $x - x_h$ are connected as neighbors to the x_h headnodes. Such a placement is the best strategy for the attacker since the impact caused by the $x - x_h$ peers is maximized due to their relative closeness to the source, i.e., a larger fraction of benign peers experience a longer service degradation till a sufficient amount of benign peers receive and start disseminating the stream. Given the fact that inferring the overlay's topology is indeed feasible [8], [9], the attacker is capable of inferring the existing headnodes to optimally place malicious peers as headnodes. Knowing the headnodes allows the attacker to place malicious nodes in their neighborhood, which also results in an effective disruption of the stream dissemination if the budget is sufficiently high.

B. Drop-chunk adversarial behavior

We now discuss the main adversarial behavior that gets executed based on the attacker's target and budget. Let M be the set of malicious peers that collaboratively execute *Drop-chunk*. When $m \in M$ receives a chunk from a neighbor, m drops the chunk. In particular, m never advertises chunks in its BM, except to the neighbor it received the chunk from. Indeed, it keeps on requesting the dropped chunks from other benign peers $b \in B$, where B is the set of benign peers. This scenario guarantees that: (a) malicious peers are less susceptible to being suspected as the requesting benign peers are not aware that m indeed received those chunks, and (b) detecting m's direct or close connection to the source, inferring the overlay's topology, is not possible, which lowers the probability of m being suspected.

Note that this behavior minimizes the detection susceptibility of malicious peers. The reason is that other BM cheating strategies result in eventually declaring a certain suspect, e.g, if m keeps on sending correct BM updates but never sends the actual chunk, honest nodes will eventually suspect m.

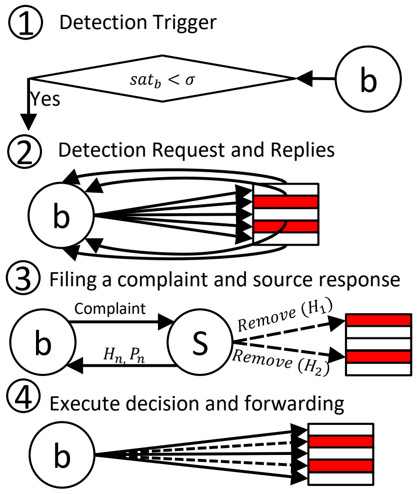
III. Detection Mechanism

In this section, we explain our detection mechanism, starting with an overview of the different steps followed by a detailed description of each step. The goal of the detection algorithm is to restore the user's satisfaction in the face of a *Drop-chunk* attack. The key idea of our method is to replace headnodes associated with peer groups of low satisfaction levels.

Throughout the section, we assume that nodes authenticate their messages using digital signatures. The source keeps track of participants' verification keys and can hence establish the authenticity of messages. In particular, malicious nodes cannot forge responses of honest peers to influence the mechanism. We assume that neighboring nodes periodically exchange messages stating that they are neighbors. These *proofs of neighborhood* are signed and contain a time stamp. In this manner, u can proof if v is (or has recently been) its neighbor.

A. Mechanism Overview

We illustrate the underlying ideas of the detection mechanism in Figure 1. When a malicious peer m performs a *Drop-chunk* attack, benign peers b are unable to immediately identify the malicious behavior. Specifically, m never sends the actual BM that represents the chunks it currently possesses, i.e., m is only requesting chunks it already has. Thus, detecting a violation in this case is not straight-forward. In particular, nodes are generally unable to identify a suspected attacker



① Detection Trigger
$sat_b < \sigma$
b
Yes
② Detection Request and Replies
b
③ Filing a complaint and source response
b
Complaint
S
H_n, P_n
Remove (H_1)
Remove (H_2)
④ Execute decision and forwarding
b

Fig. 1: Detection process for *Drop-chunk*. S denotes the source.

based only on local information. In the remainder of the section, we present a mechanism that allows nodes to collaboratively identify suspects that are subsequently removed as headnodes.

The detection consists of four steps, starting with an initial trigger of dissatisfaction at one peer and potentially terminating in replacing one or several headnodes. First, when a peer b suspects a *Drop-chunk* attack based on its local observations, b sends a *detection request* to all peers in its neighbor list. Second, each peer receiving a detection request prepares a response. Third, the initiator b decides based on the received responses if they should file a complaint to the source. If b decides to file the complaint, b sends it on behalf of the participating peers in the request. Afterwards, the source verifies the complaint, reacts accordingly, and responds to b, detailing the steps taken. The reaction of the source is either the replacement of one or several headnodes or the rejection of the removal request. Finally, b reacts based on the received response from the source and then forwards the source's reply to the other participants in the complaint, who in turn execute the same procedure.

The node b bases its decision on whether to initiate a request or forward a complaint on a number of threshold parameters, which we summarize in Table I together with various system parameters governing the attack.

TABLE I: Acronyms

Var.	Description	Var.	Description
H_n	headnodes in a complaint	P_n	potential candidates
x	no. of malicious peers	x_h	mal. headnodes
BM	buffer-map	$x - x_h$	mal. non-headnodes
sat	peer satisfaction level	σ	satisfaction threshold
t_{drop}	min. drop responses	κ	no. allowed det.

B. Detection Trigger

In order to start a detection request, the node b has to experience a low satisfaction level. The satisfaction of a peer is defined as the fraction of missed chunks, i.e., the continuity of the stream according to the $Hit/Hit + miss$ chunk ratio. In a nutshell, b starts a detection request if its satisfaction is below a satisfaction threshold σ. However, to limit the ability of malicious peers to incorrectly accuse benign peers and increase the load through false detection requests, the concrete conditions that result in a detection request from b are:

1) b's current satisfaction level sat_b is less than the predefined threshold, i.e., $sat_b < \sigma$.
2) The number of drop detection requests sent by b in the time interval t_{det} is less than κ.
3) b has not initiated or replied to any other *Drop-chunk* detection request that the source has not decided on yet.

The second and third condition guarantee that peers cannot abuse the mechanism via triggering or participating in multiple detection requests in parallel. Moreover, restricting concurrent requests for benign peers is sensible as their low satisfaction level is already noted in their reply to previous requests.

C. Processing a Detection Request

Let D denote the set of queried peers, i.e., the neighbors of the initiator b if b executes the protocol honestly. When receiving a detection request, a peer $d \in D$ hence first checks if it can participate in any more requests. If so, d replies with its sat_d and a time stamp, both signed by its private signature key. The time stamp prevents the attacker from replaying benign peers' previous (low) satisfaction levels, as only recent satisfaction levels are valid.

D. Filing and Processing a Complaint

Upon receiving a feedback from its neighbors, b decides whether to file a complaint or not. If so, the source verifies the complaint and potentially removes some of its headnodes.

Filing a complaint: b will start processing the replies once all nodes in D have replied or a time-out $t_{replies}$ occurs. We assume that the source's address is publicly known and b can send a complaint to the source directly.

b sends a complaint if the average satisfaction level indicated in the responses is below a threshold σ and at least t_{drop} peers replied to the request. More specifically, let $sat_1, \ldots, sat_z$ be the satisfaction levels expressed in the replies and sat_b be b's satisfaction level. Assuming a sufficient number of replies, b files a complaint to the source if:

$$\frac{1}{z+1}\left(sat_b + \sum_{i=1}^{z} sat_i\right) < \sigma. \tag{1}$$

Otherwise, b either starts another detection request depending on κ or waits until allowed to send another detection request.

Once b decides on filing a complaint according to the aforementioned conditions, b generates a complaint message to the source containing the IDs of all nodes in D, recent proofs of neighborhood, and the received satisfaction levels including signatures and time stamps.

The reason for requesting at least t_{drop} replies is to prevent a few malicious nodes from accusing benign headnodes. By imposing a lower bound on the requested number of replies, a considerable number of malicious nodes has to use one of their κ requests. We present an in-depth analysis on how these constraints prevent misuse in Section IV.

Processing a Complaint at the Source: The source s first verifies the content of the complaint. First, the source rejects any complaint from a node b that has already participated in κ requests. If s does not reject the complaint, s then removes any satisfaction levels without valid signatures from the complaint. Furthermore, s removes any responses from nodes that have exceeded their participation limit or are participating in two complaints at the same time.

If the remaining valid responses still indicate an average satisfaction level of less than σ, the source:

1) divides the set of peers in D into two sets H_n and P_n, where H_n is the set of headnodes peers that exist in the complaint.
2) removes all peers in H_n from its neighbor set.
3) randomly connects to another $|H_n|$ peers.
4) adds peers (excluding peers in H_n) from its neighbor list to P_n, where $P_n = NeighborList \setminus H_n$ ($NeighborList$ is the set of peers in a neighbor list).
5) sends a *Complaint Reply* to b containing H_n and P_n.

The reason for choosing random new headnodes rather than nodes participating in the complaint is to lower the incentive for complaints by malicious peers. Even if such a complaint is successful, the new headnodes are likely benign, meaning that the malicious nodes did not gain anything from initiating the request apart from slightly increasing the load.

Processing a Complaint Reply & Forwarding: Finally, when b receives the *complaint Reply* from the source, b

1) Disconnects from all peers in H_n. Note that b does not blacklist peers in H_n from its neighbor list due to the fact that those peers are not proven malicious.
2) Connects to $|H_n|$ peers from P_n, in case $|H_n| > |P_n|$, peers connect to $|P_n| + (|H_n| - |P_n|)$ random peers.

Subsequently, b forwards the complaint to the other participants, i.e., $D \setminus H_n$, who in turn execute steps 1 and 2.

E. General Notes

The detection mechanism does not aim at expelling peers from the system. Simply removing headnodes remarkably benefit the system. Indeed, the only peers that get blacklisted are those who violate the detection mechanism constraints, i.e., participating in more than a single request at a time or initiating more than κ requests. The reason for this leniency lies in the potentially high chance of removing headnodes that are benign but exhibit a low performance. In general, the main target of the detection mechanism is to enhance peers' satisfaction level while keeping peer replacements and signaling overhead low.

IV. Analysis

We focus on characterizing the behavior of malicious nodes aiming to subvert the detection mechanism to remove honest headnodes and retain malicious ones. More precisely, we show that successfully accusing a benign headnode of cheating requires that the malicious peer issuing a complaint presents a neighbor list that is either dominated by malicious peers or by benign peers with unusually low satisfaction levels. Similarly, preventing the removal of a malicious headnode requires that a high number of the neighbors are malicious.

A. Falsely Accusing Benign Headnodes

We start by considering the case that malicious nodes want to misuse the detection mechanism to remove a benign headnode. Note that there are reliable methods to identify headnodes [8], so malicious peers are likely to know if one of their neighbors is a headnode. The malicious node m initiating a request with the goal of removing one benign headnode can manipulate the set D of nodes m forwards to the source. In other words, after querying all nodes in its actual neighbor list, m might send only subset of the responses as well as responses from additional nodes to the source. If possible, m chooses these responses in such a manner that the source will remove the benign headnode. There are restrictions guiding the construction of D that m has to take into consideration:

- m should include the benign headnode it aims to remove.
- m cannot include benign nodes that are not in its actual neighbor list, as m has no valid proofs of neighborhood.
- m does not have to include all peers that are in its actual neighbor list, as there is no possibility to detect excluded neighbors short of asking all peers in the system if they are neighbors of m.
- m can include malicious peers that are not in its actual neighbor list, as these peers are willing to generate false proofs of neighborhood. Only the inclusion of malicious nodes that can participate in a *Drop-chunk* request, i.e., those that have not yet reached their limit of *Drop-chunk* request participation, is beneficial for the success of the request. Malicious peers contained in D claim that their satisfaction level is 0 to maximize the chance of removal.

When deciding on a set D, m tries to minimize the number of malicious nodes in D in order to use as few of the κ requests per node as possible. Preposition 4.1 gives a lower bound on the number of required malicious nodes.

Preposition 4.1: Let m be a malicious neighbor of a benign headnode h with satisfaction level sat_h. Assume that m has k benign neighbors $v_1, \ldots, v_k$ sorted by their satisfaction levels $sat_1 \leq sat_2 \leq \ldots \leq sat_k$. Then, to successfully remove h, m has to include at least c responses of malicious nodes, including m itself, in the set D of forwarded responses such that:

$$c = \max\Bigg(1, \qquad (2)$$
$$argmin_{c' in \mathbb{N}} \left\{ \frac{1}{t_{drop}} \left(sat_h + \sum_{i=1}^{t_{drop}-c'-1} sat_i \right) < \sigma \right\} \Bigg)$$

Proof: To remove a headnode h, there has to be a detection request containing the responses of h and $n \geq t_{drop} - 1$ nodes with satisfaction levels $s_1, \ldots, s_n$ and $\frac{1}{n+1}\left(s_h + \sum_{i=1}^{n} s_i\right) < \sigma$. The node m aims to minimize the number of involved malicious nodes c because each malicious node can only participate in κ detection requests per interval. At the same time, m has to ensure that the average satisfaction level of the involved nodes is below σ and that the request includes at least t_{drop} nodes in total. As m files the request, at least one malicious node has to be included. In other words, m solves the optimization problem of finding a minimal c and a set of integers $I \subset \{1, \dots, k\}$ such that i) $c + |I| + 1 \geq t_{drop}$, ii) $\frac{1}{c+|I|+1}\left(sat_h + \sum_{i \in I} sat_i\right) < \sigma$, and iii) $c \geq 1$. Choosing the lowest satisfaction levels indeed solves the optimization problem and results in Eq. 2. ■

For simplicity, Preposition 4.1 considers the case that only one headnode is contained in m's neighbor list. In the presence of several headnodes, m has to slightly adapt its attack strategy. If additional malicious headnodes are neighbors of m, m does not include the respective nodes in D to avoid accidentally causing the removal of malicious headnodes. In contrast, if additional benign headnodes are neighbors of m, m will include all of them in D if the detection request can be successful. If success is not possible due to the high satisfaction level of the included headnodes, m successively removes each headnode using the strategy outlined in Preposition 4.1.

B. Retaining Malicious Headnodes

Now, we consider the case that malicious nodes collude to retain one or several malicious headnodes when a benign peer initiates a detection request. All malicious nodes in the respective neighbor list will provide a satisfaction level of 1 to prevent the removal of a malicious node. We assume that malicious neighbors will try to prevent the removal of malicious headnodes even if the request can additionally result in the removal of benign headnodes. This assumption seems reasonable as the removal of a benign headnode is unlikely to lead to additional malicious headnodes, indicating that retaining existing malicious headnodes is of higher importance than removing benign headnodes. Preposition 4.2 provides the condition governing the success or failure of the *Drop-chunk* request in the face of the proposed adversarial behavior.

Preposition 4.2: Let m be a malicious headnode and b be a benign neighbor of m that initiates a detection request due to its low satisfaction level sat_b. Assume that b has k benign neighbors $v_1, \dots, v_k$ with satisfaction levels $sat_1, sat_2, \dots, sat_k$. In addition, b has y malicious neighbors, which includes the malicious headnode, and $k \geq t_{drop}$. Then the removal of m fails if and only if:

$$\frac{1}{k+y+1}\left(y + sat_b + \sum_{i=1}^{k} sat_i\right) \geq \sigma. \qquad (3)$$

Proof: The claim follows directly as all y malicious peers will set their satisfaction level to 1 and *Drop-chunk* requests with an average satisfaction of at least σ are not successful. ■

V. Evaluation

The goal of this section is to address two research questions: First, we quantify the severity of *Drop-chunk*. Second, we evaluate the proposed detection mechanism's performance in terms of effectiveness and efficiency. We start by describing the simulation model and set-up before detailing the simulation results and their interpretation for both research questions.

A. Simulation Framework, Parameters and Metrics

Our simulation framework builds on OSSim [13]. OSSim is a packet level simulator for DONet [5], a pull-based online streaming overlay. All our overlays use the network topology generator GT-ITM [14] with 1000 peers connected to 400 edge router. Furthermore, our simulation time is 500s and the presented results are averaged over 10 runs.

We differentiate between malicious and benign peers when considering their online times. We assume that malicious peers join the overlay early and do not leave before the content dissemination ends in order to maximize their impact. In contrast, benign peers join based on Pareto distribution and leave according to Lognormal distributions, as motivated by real-world measurements [15]. Benign peers can rejoin the overlay in a uniform distribution around 10s. For both case studies, the streaming rate is $400kbps$, the chunk size $2500B$ and the buffer size $30s$.

The following metrics characterize the performance.

Satisfaction sat: The satisfaction is the fraction $Hit/Hit + miss$ of chunks peers receive in time, averaged over all peers.

Avg. Loss lo: Indicates the average number of chunks that peers do not receive in time, averaged over all peers.

Detection Overhead DO: The detection overhead describes the communication overhead created by the detection mechanism. More formally, it is the ratio of messages exchanged in the system due to the detection mechanism and all signaling messages: (1) BM requests, (2) BM updates, and (3) neighboring requests, accepts and rejects.

Benign Ratio per Neighbor List $BRNL$: The benign ratio per neighbor list measures the fraction of benign peers in the source's neighbor list.

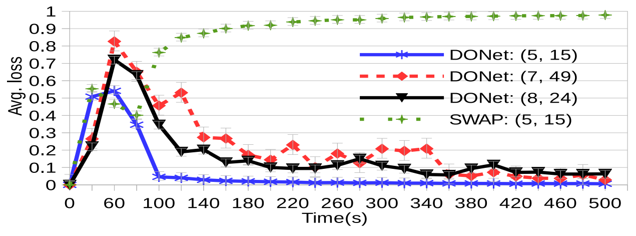
B. Case 1: Drop-chunk Severity

In this case study, we evaluate the impact of *Drop-chunk* on two different network scenarios: (1) DONet, and (2) DONet+SWAP [8]. We consider SWAP to check how replacement mechanisms of headnodes affects the attack. We use the same total number of peers but vary the attacker's budget. As malicious peers aim to occupy the closest peers to the source, the remaining size of the overlay is not a factor on the impact of the *Drop-chunk* attack.

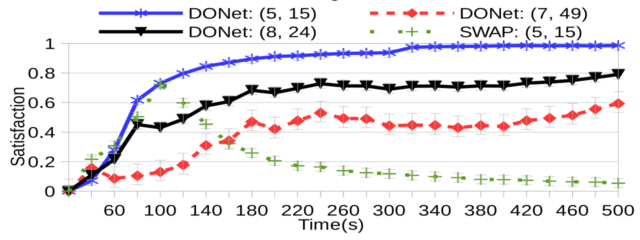
Given the source's neighbor list size $LS = 10$, we choose the following combinations for the attackers budget $(x_h, x - x_h)$: $(10, 0), (5, 15), (7, 49), (8, 24)$. Here, $x - x_h = 49$ denotes that 7 malicious peers are connected to each of the 7 malicious headnodes. We start with analyzing the attack's impact on DONet and then we evaluate the resilience of SWAP to the attack.

Figure 2a displays the average chunk loss ratio. Unsurprisingly, the average loss is 100% when $x_h = LS = 10$, which means that the source's LS is utterly saturated with malicious headnodes, i.e., no chunks are transmitted to the rest of the overlay. Thus, the average peer satisfaction is always 0%.

If $x_h < 10$, the average loss initially reaches up to 82% for $(x_h, x - x_h) = (7, 49)$, for $(x_h, x - x_h) = (5, 15)$, the loss ratio is 54% and 73% for $(x_h, x - x_h) = (8, 24)$, as

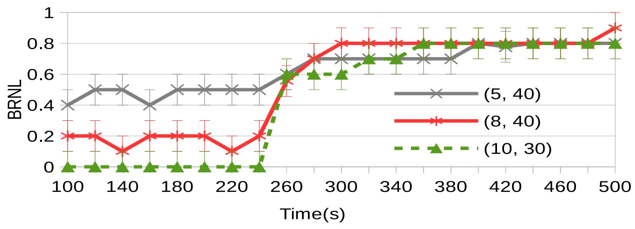


(a) Average loss

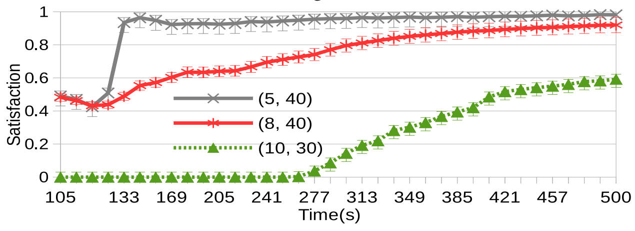


(b) Average peer satisfaction

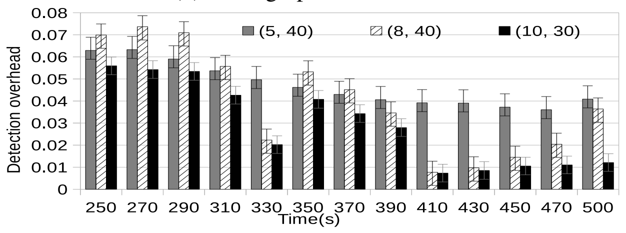
Fig. 2: Attack's impact on DONet



(a) Average BRNL



(b) Average peer satisfaction



(c) Detection overhead

Fig. 3: Detection mechanism performance

shown in Figure 2a. If x_h or $x - x_h$ increases, benign peers experience severe service degradation for a longer time period. Benign peers close to the source suffer from overload, leading to a high ratio of missed chunks. Nevertheless, the loss ratio decreases once a fraction of benign peers are able to serve the rest of the overlay.

Figure 2b presents the average peer satisfaction level sat. As a consequence of experiencing high chunk loss rate, higher values of x_h and $x - x_h$ result in lower peer satisfaction over time, where benign peers at $(x_h, x - x_h) = (5, 15)$ restore their satisfaction level at approximately 340s, which is earlier than at $(x_h, x - x_h) = (7, 49)$ and $(x_h, x - x_h) = (8, 24)$.

Now, we analyze the attack's impact while SWAP is operating. During SWAP, peers nominate new headnodes and forward these nominations to the source. Malicious peers abuse the mechanism by nominating other malicious peers at each nomination round. Moreover, malicious peers connected to benign headnodes are eventually nominated to the source and thus can occupy the source's neighbor list LS.

As shown in Figure 2a, comparing the same values at $(x_h, x - x_h) = (5, 15)$ for both DONet and SWAP show that the impact of the attack is more significant if SWAP is active. Before the source's LS is saturated with malicious peers at $t = 80s$, the average loss is in fact decreasing, However, as soon as malicious peers control the neighbor list, the average loss increases up to 97%. For the same reason, the satisfaction level of benign peers eventually decreases to 6%, which highlights the unsuitability of SWAP against our proposed attack.

C. Case 2: Detection Mechanism Performance

We now evaluate the performance of the detection mechanism. Benign peers execute the detection mechanism as described in Section III whereas malicious peers aim to misuse the mechanism. More precisely, malicious peers reply with a satisfaction level of 0 if the complaint might remove a benign peer and 1 if it might remove a malicious peer.

In order to do so, we chose $(x_h, x - x_h) \in \{(5, 40), (8, 40), (10, 30)\}$ to assess the performance of the mechanism in severe attack conditions. The satisfaction threshold σ is set to 0.95 to measure if peers are able to fully restore their satisfaction level when the detection mechanism is operating. The detection mechanism is effective starting $t = 250s$ to allow for a reasonable amount of peers to join the overlay to adequately assess the efficiency of the mechanism. In this scenario, every peer is allowed to initiate $\kappa = 10$ detection requests for $t_{det} = 500s$, and the minimum number of responses to generate a complaint is $t_{drop} = 3$. We discuss the effect of varying those parameters later on.

As depicted in Figure 3a, we observe an increase in the benign headnodes ratio in the source's neighbor list after the detection mechanism starts operating at $t = 250s$. For instance, if $(x_h, x - x_h) = (5, 40)$, the source successfully attains 80% benign headnodes due to the detection mechanism. For $(x_h, x - x_h) = (8, 40)$, the $BRNL$ ratio increases up to 90-100%, which reflects the efficiency of the detection mechanism in replacing malicious headnodes to restore peers' satisfaction levels. Even if the source is initially only connected to malicious headnodes, i.e., $x_h = 10$, the detection mechanism is capable of restoring a $BRNL$ to approximately 80%.

Figure 3b illustrates the average restored peer satisfaction level when the detection mechanism is active. For all considered attack budgets, the average satisfaction level quickly increases to 95-100% for $(x_h, x - x_h) = (5, 40), (8, 40)$. For $x_h = 10$, the average satisfaction increase from 0% to almost 60% in a time span of 250s. The reason of the quick increase is that the number of initial detection requests sent to the source results in replacing a high fraction of the source's headnodes. Moreover, malicious peers are unable to misuse the mechanism, as indicated by the absence of degradation in satisfaction levels.

Figure 3c depicts the average detection overhead induced by our mechanism. The maximum overhead due to the detection mechanism is 8% for all scenarios. As peers are eventually satisfied, the number of detection requests initiated decreases and the overhead decreases to 4% at $t = 500s$, i.e., peers stop invoking the mechanism. Moreover, the maximum number of detection requests that can be initiated is dependent on κ, which is set to 10 in this scenario. Thus, smaller values of κ result in lower overhead. In fact, varying t_{drop} between 3, 4 and 5 has little impact on the detection performance, indicating that nodes receive sufficient replies.

In summary, our simulation study highlights the effectiveness efficiency of the detection mechanism against *Drop-chunk* attacks, even in the presence of an attacker that initially controls the majority of nodes close to the source.

VI. Related Work

We overview the prominent existing work on attacks and their detection in the context of P2P streaming systems. Most prior work has considered three attack types: (i) pollution attacks, i.e., flooding the overlay with arbitrary content and claiming it to be relevant chunks, (ii) free riding, i.e., participating in the overlay without contributing, and iii) cheating attacks, i.e., maliciously dropping packets or manipulating buffer-maps.

Pollution attacks are one of the most common attacks [16]. Strategies such as network coding [17] and [18] can effectively mitigate these attacks.

In contrast, the main approach to counter free riding are incentives [6], [19], i.e., rewarding peers that distribute the stream to others. However, these strategies are only effective for peers that aim to minimize their level of participating.

Cheating attacks are severe DoS attacks, performed to maximize the damage to the overlay and preventing peers from downloading the stream. *Antiliar* is a general defense mechanism against a diverse set of attacks, including dropping and buffer map manipulation [7]. Mainly, *Antiliar* tracks peers behaviors in a secure progress log and thus, detecting misbehaving peers by identifying irregularities in the log. While highly effective, *Antiliar* relies on expensive cryptographic operations that are unsuitable for devices with low CPU resources. Moreover, *Antiliar* uses a central entity to review the logs, creating additional security and privacy problems.

An alternative decentralized approach [10] relies on redundancy by enforcing diversity when requesting chunks. In this manner, the attacker has to control a higher fraction of nodes to achieve any severe damage by cheating. The work focuses on attacks on headnodes yet assumes an external attacker that can take over arbitrary nodes at will. In this context, the idea of swapping headnodes frequently to mitigate the impact of the attacker's control significantly decrease the attack severity [8]. As shown in Section V, internal attackers can undermine the swapping protocol and gain the position of headnodes.

VII. Conclusion & Future Work

In this work[1], we focus on the class of internal inference attacks for pull-based overlays. The attacker conducts a BM cheating attack after placing malicious peers as headnodes. We show that the attack severity significantly increases the chunk loss ratio, accompanied by low satisfaction level experienced by benign peers. As a countermeasure, we propose a detection mechanism where peers are able to collaboratively file a complaint to the source when their average aggregated satisfaction drops below a certain threshold so the source can replace suspicious headnodes.

Our simulations show that the detection mechanism is capable of restoring 95-100% of peers satisfaction level while removing 80-90% of malicious headnodes from its neighbor list while inducing a low overhead of approximately 8%. As an ongoing work, we focus on evaluating the resilience of our approach against various BM cheating strategies and integrating anonymous monitoring for proactive defense.

References

[1] Emule Digital Content. http://emule.com.

[2] Sasi, L. et al. A Survey on Peer to Peer Video Streaming Systems. *Proc. IJRECE*, 3, 2014.

[3] Zhang, J. et al. Modeling and Performance Analysis of Pull-based Live Streaming Schemes in Peer-to-Peer Network. *Computer Communications*, 40:22–32, 2014.

[4] Cui, Y. et al. Impact of Buffer Map Cheating on the Streaming Quality in DONet. In *Proc. ICCS*, pages 817–824, 2007.

[5] Zhang, X. et al. CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-peer Live Media Streaming. In *Proc. INFOCOM*, pages 2102–2111, 2005.

[6] Habib, A. and Chuang, J. Incentive mechanism for peer-to-peer media streaming. In *Proc. IWQOS*, pages 171–180, 2004.

[7] So, K. and Reeves, S. Antiliar: Defending Against Cheating Attacks in Mesh Based Streaming. In *Proc. P2P*, pages 115–125, 2012.

[8] Nguyen, G. et al. SWAP: Protecting pull-based P2P video streaming systems from inference attacks. In *Proc. WoWMoM*, pages 1–9, 2016.

[9] Nguyen, G. et al. RBCS: A resilient Backbone Construction Scheme for Hybrid Peer-To-Peer Streaming. In *Proc. LCN*, pages 261–269, 2015.

[10] Nguyen, G. et al. On the Resilience of Pull-based p2p Streaming Systems Against DoS Attacks. In *Proc. SSS*, pages 33–47, 2014.

[11] Huang, Y. et al. Challenges, Design and Analysis of a Large-scale P2P-vod System. In *Proc. ACM SIGCOMM*, pages 375–388, 2008.

[12] Vu, L. et al. Measurement and Modeling of a Large-scale Overlay for Multimedia Streaming. In *Proc. QSHINE*, pages 3:1–3:7, 2007.

[13] Nguyen, G. et al. Ossim: A Generic Simulation Framework for Overlay Streaming. In *Proc. SCSC*, pages 30:1–30:8, 2013.

[14] Zegura, E. et al. How to model an internetwork. In *Proc. INFOCOM*, pages 594–602, 1996.

[15] Veloso, E. et al. A hierarchical characterization of a live streaming media workload. *ACM Transactions on Networking*, 14:133–146, 2006.

[16] Gkortsilas. I. et. al. Detecting and Isolating Pollution Attacks in Peer-to-peer VoD Systems. In *Proc. EuCNC*, pages 340–344, 2016.

[17] Fiandrotti, A. et al. Simple countermeasures to mitigate the effect of pollution attack in network coding-based peer-to-peer live streaming. *IEEE Transactions on Multimedia*, 17:562–573, 2015.

[18] Kang, X. et al. A Trust-based Pollution Attack Prevention Scheme in Peer-to-peer Streaming Networks. *Computer Networks*, 72:62–73, 2014.

[19] D. Li. et al. Defending Against Buffer Map Cheating in DONet-Like P2P Streaming. In *IEEE Trans. Multimedia*, pages 535–542, 2009.

[1]Research supported in part by EC H2020 CIPSEC GA #700378 and BMBF TUD-CRISP