An SDN approach to detect targeted attacks in P2P fully connected overlays

Cristóbal Medina-López[1] · **L. G. Casado**[1] · **Vicente González-Ruiz**[1] · **Yuansong Qiao**[2]

Abstract
Pollution attacks are one of the major concerns facing P2P networks. They have a tremendous impact on push-based fully connected overlays, in which each peer receives an exclusive chunk from the source and is also the only one responsible for relaying it to the rest of the peers. In this study, we propose a novel technique to identify and expel malicious peers which involves using trusted peers, software-defined networking (SDN) and proactive moving target defense. Experiments to obtain the accuracy and effectiveness of the implemented methods, as well as an analysis of the performance concerns, were carried out through simulation using a Mininet network emulator. The experiments demonstrate the feasibility of our proposal, which provides high rates of detection, not only in pure SDN environments but also in mixed ones.

Keywords SDN · Pollution attacks · Trusted peers · Peer-to-peer · P2PSP

1 Introduction

Client–Server (CS) architectures for streaming multimedia content on the Internet experience scalability issues when a large number of clients try to access the same media content. This problem worsens when live broadcasts are involved. To address these scalability problems, many peer-to-peer (P2P) protocols have been developed over the last two decades.

Compared to CS systems, in P2P overlays, the peers contribute with their upload bandwidth, allowing the transmission of the stream to more receivers in real time. Moreover, this approach brings some interesting benefits, such as traffic reduction or a decrease in latency, particularly within ISP (Internet service provider) networks and last-mile scenarios.

Unfortunately, the contribution of the peers to the overlay makes it vulnerable to attacks, especially in the case of push-based fully connected overlays, where each peer receives an exclusive chunk from the source and is the only one responsible for relaying it to the rest of peers. Two of the most common attacks are pollution and free-riding attacks, and they become much more difficult to mitigate when they are selective, i.e. they targeted certain peers but not others. Such actions make detection more difficult to achieve because Malicious Peers (MPs) try not to be detected by trusted peers [1] (TPs) or trust management systems. For example, the attackers can collaborate to detect who TPs are, thereby avoiding attacking them by conducting a selective attack.

To manage these problems, trust management systems, or incentive mechanisms have been proposed ever since P2P networks became popular [1–4]. However, the performance of such techniques in push-based fully connected overlays is not ideal as most of them allow malicious behaviors in the team which are under a certain threshold. In some cases, they even make wrong decisions, resulting in Honest Peers (HPs) being unfairly expelled from the team.

The sender needs to know the destination endpoints (EPs) to perform communication on the Internet. Apart from servers load balancing, DHCP, etc., endpoints usually determine univocally a final device. This helps MPs to estimate how safe an attack based on previous information about the

✉ Cristóbal Medina-López
cristobalmedina@ual.es

L. G. Casado
leo@ual.es

Vicente González-Ruiz
vruiz@ual.es

Yuansong Qiao
ysqiao@research.ait.ie

[1] University of Almería (CeiA3), Almería, Spain

[2] Athlone Institute of Technology, Athlone, Ireland

[1] TPs are able to verify whether a message is legitimate or missing, and they will report any bad behavior if necessary.

endpoint is. A new architecture, called software-defined networking (SDN), aiming to facilitate network management as well as enable efficient programming of network configuration is becoming increasingly more popular. In this context, the programmable feature of SDN makes it possible to dynamically modify the destination End Point (EP, i.e., the IP address and port) in the packets in order to implement a moving target defense (MTD). For this reason, we herein study a technique based on SDN and MTD to identify and expel MPs from the overlay in a short period of time and avoiding incorrect decisions, i.e., without expelling honest peers.

The main contributions of this research are the following:

- Study a method for detecting and expelling malicious peers in a P2P push-based fully connected overlay in a deterministic way, avoiding the *false positive* and *false negative* results observed in other methods such as peer polling or trust management systems.
- Perform experiments and analyze the performance of the proposed technique using the peer-to-peer straightforward protocol (P2PSP) simulator.
- Study the feasibility of applying our proposal to a mixed network using SDN and non-SDN devices.

This paper is organized as follows. Section 2 summarizes the P2PSP protocol. A review of the related work in targeted attacks and MTD topics is shown in Sect. 3. Section 4 describes our proposal for discovering and expelling malicious peers by using an SDN approach. Several experiments are carried out in Sect. 5 to obtain the accuracy and effectiveness of the proposal used in conjunction with the P2PSP protocol. Finally, our conclusions are shown in Sect. 6.

2 Peer-to-peer straightforward protocol

In order to specifically explain the problem we are addressing and the proposed solution, a brief description of the peer-to-Peer straightforward protocol (P2PSP) is introduced in this section. P2PSP was proposed as an application-layer protocol that provides real-time broadcasting, also known as application layer multicast (ALM), of a media stream on the Internet. It has a modular design implementing different functionalities, which allow us, among other things, to deploy a hybrid architecture with CDN, and P2PSP in the "last-mile". One of the hardest problems to tackle in P2PSP (and, by extension, in push-based fully connected protocols) is the malicious behavior of some peers. Although we experimented with some solutions to mitigate these problems [5,6], they were not fully neutralized. In this study, we take advantage of the SDN technology to eradicate targeted attacks.

2.1 Description

P2PSP [7] deploys *push-based mesh-shaped overlays*, and is specifically designed to minimize the latency of live media streaming, particularly in local environments. Therefore, hybrid CS/P2P structures can be used to deploy massive systems with P2PSP. For instance, the media could be first transmitted at the backbone level over a CDN, and P2PSP could be used for the last-mile links. In this way, the solution shows a better bandwidth consumption and it is more ISP-friendly than a complete P2P approach. Moreover, this allows for a low latency because peers inside a P2PSP team (a single P2PSP overlay) are close to each other.

Some key features of P2PSP are:

- *Content agnostic* The protocol does not analyze the transmitted information and does not depend on a specific media format.
- *Modular architecture* It is structured in different modules, each one providing a different functionality. The selection of the modules depends on the requirements of the target system.
- *Low requirements* The most basic modules can run on resource-constrained hardware.
- *IP multicast support* IP Multicast can be used when available.
- *Private network support* Peers can participate from within private networks, even when they are behind most NATs, including certain types of symmetric NATs.
- *Full-connected topology* When available, all nodes push data to the rest of the nodes in 1-overlay-hop, minimizing the latency.
- *I/O ratio close to 1* In full-connected scenarios, all nodes send and receive the same amount of data.
- *CDN friendly* It is compatible with the CS model, allowing a hybrid architecture.

2.2 Node roles

P2PSP defines the following entities (see Fig. 1):

- A **Source** It produces the stream. It is usually a streaming HTTP server hosted by the content provider which produces the content or relays it from other sources. Supposedly, the source controls the transmission bit-rate for the entire team. P2PSP does not understand the stream content, and, therefore, it is not able to perform a data-flow control.
- A **Splitter** (S) S receives the stream from the source, splits it into chunks of data and sends them to the peers.
- One or more **Peers** (P) Each peer receives chunks from S and other peers, sorts them into a buffer to reconstruct the original stream and sends them to the player. All chunks

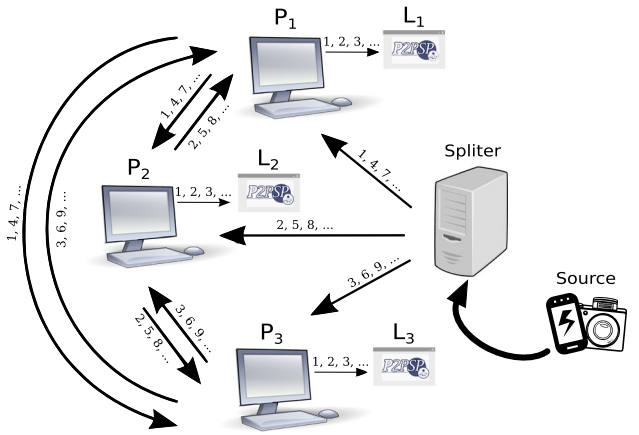

Fig. 1 A typical P2PSP team running the data broadcasting set of rules. The numbers on the arrows represent the chunks that are being transmitted

received from S are relayed to other peers, following a predefined set of rules.

- One or more **Monitor Peers** (P) They are similar to regular peers but with the extra functionality of reporting which chunks are lost to S. They can act as TPs.
- One or more **Players** (L) They are in charge of decoding and playing the media. Each peer can feed several players simultaneously.

2.3 Data broadcasting set of rules

Data broadcasting set of rules (DBS) describes the minimal set of rules needed for P2P broadcasting over the Internet. In DBS, S sends the chunks to the peers using a round-robin scheduling scheme, and each peer relays the last chunk that has been received from S to each other peer. In order to avoid network congestion, peers relay a chunk upon reception of another chunk from another peer or S. However, upon a new chunk reception from S, the previous chunk is relayed in burst mode to pending peers.

Every node of the team knows the EP of the rest of the peers and has access to the peer list used by S to send the chunks to peers, which in turn is used by peers to forward the chunks received from S to each other peer. When an incoming peer P_x is accepted by S to join the team, S sends it the current list and adds P_x to the list. In order to accelerate the inclusion of P_x in the team, P_x sends a [`hello`] message to every other peer as soon as the other peer's EP is received as a new element of the list. When P_x leaves the team, it sends a [`goodbye`] message to the rest of the team. Then, S and the rest of the peers that remain in the team stop sending chunks to P_x.

In P2PSP, every peer must retransmit the same amount of data that it receives. Thus, the overlay is fair and it is easier to recognize free-riders. A peer that cannot fulfill this rule must leave the team. In order to achieve this requirement, each peer P_i assigns a counter to every other peer P_j of the team. When a chunk is sent to P_j, its counter $|P_j|$ is increased, and when a chunk is received from P_j, $|P_j|$ is decreased. If $|P_j|$ reaches a threshold, P_j is deleted from the list of peers of P_i and it will not be served anymore by P_i. Using this procedure, peers P_k leaving the team that failed to send the [`goodbye`] messages are automatically treated as selfish peers by the rest of the team and removed from their list of peers. S detects this situation because the monitors of the team report the chunks lost to S, which remembers the receiver of each chunk it has sent. For each peer, S counts the number of lost chunks. A peer is removed from the list of S when the counter for that peer reaches a threshold in a given interval of time. Moreover, lost chunks are retransmitted by S to the existing team.

3 Related work

3.1 Targeted attacks in P2P networks

Pollution attacks are based on relaying chunks of the stream with polluted information instead of correct chunks. Versions differ in how the receivers are selected (random, selective or all) and whether the malicious peers collaborate or not to pollute chunks in order to not be detected [5]. The effects of the pollution attacks on P2P video streaming systems are significant [8]. Such repercussions have brought about the advent of several techniques in the literature, most of which are focused on pull-based systems, where peers ask for chunks from other peers and maintain the peers that behave correctly with a better performance, i.e. send correct chunks with bigger throughput. Therefore, in pull-based systems malicious peers are easily detected and isolated. On the contrary, in push-based overlays peers send chunks to other peers without explicit requests. The advantage of push-based systems is the reduction of bandwidth usage due to the lack of requests. However, studies on overcoming pollution attacks for fully connected push-based overlays are scant. The main problem with this type of pollution attack is that receiving peers do not usually select the sending peers and therefore, it is more sensitive to pollution and free-riding attacks. Both types of attacks have the same impact and can be treated in the same way.

Peer polling and trust management systems are frequently used solutions to deal with pollution attacks [1,2]. Unfortunately, they can produce false positive and false negative results when a targeted attack is carried out.

Although non-repudiation systems are a good approach, since they are reliable in making decisions about expulsion, a serious issue arises when they are applied to P2P systems: a trusted third party (TTP) is usually implicated in the process, which does not satisfy one of the main requirements

of the P2P systems: scalability. In the literature, we can find some approaches that do not use TTPs, for example [9], but these are based on the premise that MPs are interested in the content, an assumption that does not necessarily have to be true.

Another popular solution is the use of incentive mechanisms, as they make sure that only fruitful connections are kept. Unfortunately, this approach is not feasible in a P2P push-based fully connected overlay because MPs do not need the chunks from others in order to pollute or to refuse to send the chunks they relay.

In recent years, we have studied several techniques for fighting against attacks in P2P push-based fully connected overlays. As we concluded in [6], the combination of a collaborative and a selective attack has a severe impact on a P2P push-based fully connected system. In fact, we proposed a simple mechanism which introduces TPs into the team and tries to hide them with the objective of detecting and expelling the MPs. We realized that a defense strategy using only TPs is not suitable when the number of MPs is large. This is because we are dealing with a multi-objective optimization problem and each decision made by MPs may have a TP's countermeasure. For instance, under a collaborative-selective attack, when a TP is discovered, it will never be attacked again. So, the effectivity of a TP disappears. We proposed a novel solution based on Shamir's secret sharing to force peers to relay unpolluted content to HPs in order not to be expelled [5]. In that study, our motto was simple: *if you want to remain in the team you must show good behavior with at least t peers*. After analyzing our proposal, we concluded that, under the premise that there are fewer MPs than HPs, the most severe attack is fully mitigated. However, there are still other attacks that are not detected using this strategy.

One obvious option to improve the detection performance is to increase the number of TPs. However, we realized that the main problem in previous models is *the static nature of the system*. In the first model, MPs are able to perform a collaborative attack in a selective way to scan the network looking for TPs, identify them, and behave correctly with them but not with others. In this way, they remain in the team without being expelled. In the second model, MPs can survive in the team indefinitely as long as they behave correctly with a number of peers greater than or equal to Shamir's threshold.

Nowadays, thanks to technologies such as SDN, it is possible to design new techniques in order to make the network more dynamic. A good example of this is the concept of MTD.

3.2 Moving target defense

Recent research on the proactive defense concept has led to two major mechanisms: (i) moving target defense, which increases the complexity, diversity, and randomness of the cyber systems to disrupt adversaries' reconnaissance and attack planning, and (ii) cyber deception, which provides plausible-looking yet misleading information to deceive attackers [10]. Moving Target Defense (MTD) is a research hotspot in the field of network security [11], and a survey of MTD can be found in [12]. Our approach can be classified as Moving Target Network Defense (MTND) with IP address and port mutation [13]. Network Address Space Randomization (NASR) force hosts to change their IP addresses frequently [14]. Instead of performing a random change of peers' IPs, a transparent moving target defense using SDN is presented in [15]. In a similar way, our model is based on a random change of target IPs among peers by scrambling peers' EPs in SDN devices. In this way, the original P2PSP protocol is unaltered. An introduction to network address shuffling can be found in [16].

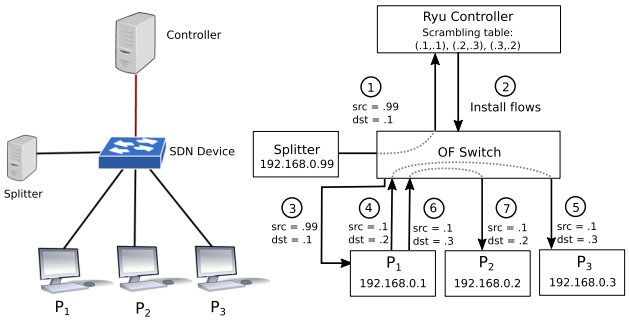

Fig. 2 A simple version of the network architecture with only one SDN device. The communication scheme is also shown. The ports of the EPs have been ignored in this example for the sake of simplicity. Note that source (src) and destination (dst) IPs are represented in the figure with the last octet in decimal format

4 Studied solution

Keeping our previous strategy for combating pollution/free-riding attacks nearly intact by using TPs [6], we introduce an SDN controller algorithm which is applied only to the UDP traffic involved in the P2PSP protocol.

4.1 Detection method

As shown in Sect. 3, the main weakness of adding TPs in a P2P team is that TPs could be discovered by MPs and, as a result, they could perform targeted attacks, thereby avoiding the pollution of chunks whose destination is a TP. Therefore, a simple way to prevent such as situation is hiding the destination EPs from the entire team (excluding S). However, in a P2P system, every peer needs to communicate with the rest of the peers directly, so they have to send packets to their specific EPs.

To achieve the goal of changing the destination EPs, we have designed and implemented the pseudocode shown in Algorithm 1.[2] This algorithm is located in the SDN controller as a Ryu app [17], and consists of exchanging the destination EP of messages sent by peers using an association table. The entries of this table are indexed by the destination EP to which a peer intends to send a message, and they contain the actual EP of the peer that will receive it. An example is presented as follows (see Fig. 2):

(1) S sends a chunk to P_1, the first peer in the team. The OF (OpenFlow) switch redirects the packet to the Ryu controller as it has no rule in charge of managing this. The controller takes a reactive approach, i.e., it analyzes all messages from S to know when a new round will arise (a round starts when S sends a message to the first peer in the list of peers in the team).
(2) The Ryu controller, which runs our P2PSP Ryu app (Algorithm 1), generates a table, the so-called *scrambling table*, which associates the original and actual destination EPs of the peers. This assignment is randomized in each round. The scrambling table is sent in the form of rules (allowed flows) to the SDN device using the OpenFlow protocol [18].
(3) After installing the new flows, the chunk is sent to the destination P_1.
(4) P_1 sends a chunk to the first peer in its list of peers. Let us assume that it is P_2. Therefore, the source of the message is the IP address 192.168.0.1, and the destination IP is 192.168.0.2.[3]
(5) When the message reaches the SDN device, the destination is changed and then redirected to the new EP. In the example of the Fig. 2 , P_3 (192.168.0.3) is the actual destination address and P_3 receives the message.
(6) P_1 sends a chunk to the second peer in its list of peers, which is P3 according to our example. Therefore, the source of the message is the IP address 192.168.0.1, and the destination IP is 192.168.0.3.
(7) As in Step 5, when the message reaches the SDN device, it redirects the message following the rules received from the controller. Then, the original destination is changed to a different destination. In this case, P_2 (192.168.0.2) is the destination which receives the message.

The random shuffle applied to the EPs of the peers in the team can cause the new destination to match the source of the message. In this situation, the controller makes a new change to send the message to another destination different from that of the source. For instance, according to the scrambling table in Fig. 2, when P_2 sends a message to P_3, the new destination is P_2 ((.3,.2) in the scrambling table), but this would cause the loss of the message. In this case, the controller uses the element in the scrambling table that has the original origin (.2), which is (.2,.3). Thus, the final destination is P_3. This does not cause duplicates at reception.

[2] The full version of the code is available as open source at https://github.com/P2PSP/SDN-P2P.

[3] In this discussion, the ports of the EPs have been ignored for the sake of simplicity.

Algorithm 1 P2PSP Ryu app algorithm

INPUT: peer_list, packet, splitter

```
1: scrambling_table = dict(zip(peer_list, peer_list))          #Create a
   scrambling table for associating original EPs with new destinations
   EPs
2: if P2PSP UDP packet then
3:     if splitter's packet and new round then
4:         clean_flows()                          #Remove rules from devices
5:         shuffle(scrambling_table)              #Assign a new EP mapping
6:     else if packet destination in scrambling_table then
7:         store mapping for IP to Mac and Mac to Port
8:         if scrambling_table[destination] is myself then
9:             destination = myself
10:        new_destination = scrambling_table[destination]
11:        add_Flow(new_destination)    #Add new rules for the device
12: send_PacketOut()                                  #Send modified packet
```

4.2 Vulnerability analysis

Since peers can only decide to send the chunks or not and, in the cases where they are sent, whether they are polluted or not, it is impossible for MPs to bypass the system using these attacks.

In the specific case of the P2PSP protocol, one of the rules in DBS says that "peers stop communicating with a peer if they do not receive anything from it". When using an SDN, this rule constitutes a weakness because an attacker could know the real peer who received its packet. For instance, if an MP attacks peer P_i, and in the next round P_j does not send anything to the MP, it could know that the EP assigned in its list of peers for P_i in the previous round was actually mapped to the EP of P_j. A similar procedure could be used by a group of MPs to detect whether or not they are behind an SDN device. With the aim of making MPs unable to perform target predictions, we opted to remove said rule from the protocol and each HP continues forwarding chunks to all members of the team, even when it is attacked.

We could imagine a challenging scenario in which the number of MPs is half of the team, and each MP attacks just one peer in a collaborative way (in which attackers share information about the attack). Thus, they can know the current scrambled EP for the TPs: which are the EPs used by the expelled peers. However, as soon as a new random scrambling is done, all TPs are hidden again.

To determine the viability of our approach we evaluate the following security MTD properties of our system. For this purpose, we use the properties identified in [19,20]:

- *Unpredictability* As previously stated, our approach includes the use of a random number generator to shuffle the EPs in a transparent way for peers. Thus, MPs cannot guess the real destination EP in a new round scrambled by themselves.
- *Vastness* The scrambling is performed on a current set of peers' EPs. Exhausting the EP (or even the IP) space is a general attack that will not affect the P2P protocol for the current set of peers.
- *Periodicity* The scrambling period parameter guarantees the periodicity.
- *Uniqueness* Although peers do not know who the real destination is, the controller knows this information all along. Therefore, all peers in the team are uniquely identified by their real EPs.
- *Revocability* All flows in our system are specific to the destination. So, it is possible to modify individual records.
- *Availability* Our approach does not introduce new availability constraints and thus meets the availability property.
- *Distinguishability* Trusted peers are in charge of distinguishing HPs from MPs and expelling the latter.

4.3 Probability of detection

Let us suppose we are dealing with a team of 10 peers, in which one is a TP and one is an MP in a managed network, and the scrambling is performed before every round. MP only pollutes one chunk or does not send a chunk in each round. The probability that the MP attacks the TP in the first round is $p = \frac{1}{9} = 0.111$. However, what is the probability of TP being attacked in the second round? For that to happen, two events must occur: i) The MP does not attack the TP in the first round, and ii) the MP does attack the TP in the second round. Therefore, the probability is $p = \frac{8}{9} \times \frac{1}{9} = 0.098$. Based on this information, the probability that the MP attacks the TP in round r in a team of size $Z > 2$ is

$$p = \left(\frac{Z-2}{Z-1}\right)^{(r-1)} \times \frac{1}{Z-1}. \tag{1}$$

However, we are more interested in knowing the probability of detecting an MP after N rounds than precisely in round r. Therefore, the expression of the probability we are interested in is

$$p = \sum_{r=1}^{N} \left(\left(\frac{Z-2}{Z-1}\right)^{(r-1)} \times \frac{1}{Z-1} \right). \tag{2}$$

4.4 Cost of performing scrambling

There is certain essential information that must be known in order to measure the impact of implementing this solution in an SDN environment. It could be defined in terms of network traffic, such as the exchange of messages between the controller and switches, or the number of flows to be installed. It could also be defined in terms of detection cost. Both options will depend on how many rounds of scrambling take place, yet this will be different in each scenario.

As for the network cost, this can be reduced by decreasing the number of rounds in which a new scrambling takes place. However, this action increases the detection time because the higher the number of rounds to redo the scrambling, the later the detection of the MPs.

Therefore, the additional costs added by this solution are:

- Traffic of chunks from S to the controller in order to monitor the round status (caused by a reactive approach). It could be optimized by sending direct orders from S to the controller only in each new round (proactive behavior).
- Flows cleaning and re-installation, which will depend on the scrambling period, one or several rounds.

Every time that a scrambling is performed, three related tasks are required, which in turn affect the overhead:

1. Cleaning the flows in all SDN devices. To do so, the controller has to send a *OFPT_FLOW_MOD* message [21] to all devices in order to remove every flow from their tables. The process of sending such a packet entails an overhead of 48 bytes for each SDN device.
2. Scrambling EPs in the controller dictionary. This involves a cost related to the CPU usage, which will depend on the number of EPs to scramble and, of course, on the used algorithm used to generate randomness. This overhead can be considered negligible for present-day SDN controllers.
3. Installing new flows in all SDN devices. A new *OFPT_FLOW_MOD* packet is sent to each SDN device where peers are connected. It is also necessary to send flows to create connections among SDN devices. Therefore, the overhead in this step is the sum of computing the following expression for each SDN device in our managed network: 48 bytes $\times$ (connected peers + SDN connected devices).

4.4.1 Modifying the scrambling period

One of the main concerns in our approach is related to the increase in the network traffic and the number of flows installed in the SDN devices. Both of them are closely related to the scrambling period parameter.

The probabilities of detection are reduced as the scrambling period is increased. These results can be modeled by adding the scrambling period parameter s to the expression in Eq. (2), resulting in

$$p' = \sum_{r=1}^{N} \left(\left(\frac{Z-2}{Z-1} \right)^{(\lceil \frac{r}{s} \rceil - 1)} \times \frac{0^{(r \mod s)}}{Z-1} \right). \quad (3)$$

In this way, the network manager can decide on a trade-off between the scrambling period r and detection efficiency p'.

4.4.2 Minimizing costs

In the current implementation, we take an *SDN reactive* approach to managing flow entries installation, which means that if there is not a match for the flow in the SDN device, it creates a *packet-in* packet [21] and sends it to the controller for further instructions. The controller analyzes the *packet-in* and sends the appropriate flow entries to the device. Therefore, if the scrambling period is one (changes in each round), the flow entries are valid only for one round and have to be updated for each round. In cases where the audience is known beforehand, for instance in prepaid pay-per-view events, we could take an *SDN proactive* approach. Rather than reacting to a packet, it consists of populating the flow tables from the controller to all SDN devices for any future traffic that matches the flows among all possible peers. If some predicted peers are not connected, this solution only affects the overhead because peers send chunks to non-existent peers, but the condition that all existing peers receive the stream is fulfilled. Moreover, we could adopt a number of controller location techniques for scalability and traffic optimization reasons [22].

4.5 Mixed network using SDN and non-SDN devices

Our approach was designed for proper operation in a network architecture where all devices are equipped with SDN technology. However, it could prove to be of great use under a network which mixes traditional and SDN devices. To accomplish this, some decisions must be made about where the scrambling is performed.

Assuming a network with SDN and non-SDN devices we must consider different scenarios:

– SDN devices are in the core of the networks, i.e., no peer is connected directly to them. Under this scenario, it is very risky to carry out an EP scrambling because we must first know the routing table for every device in the network which is traversed by chunks in order to make sure that all peers receive all chunks. Unfortunately, this is something that cannot be guaranteed.

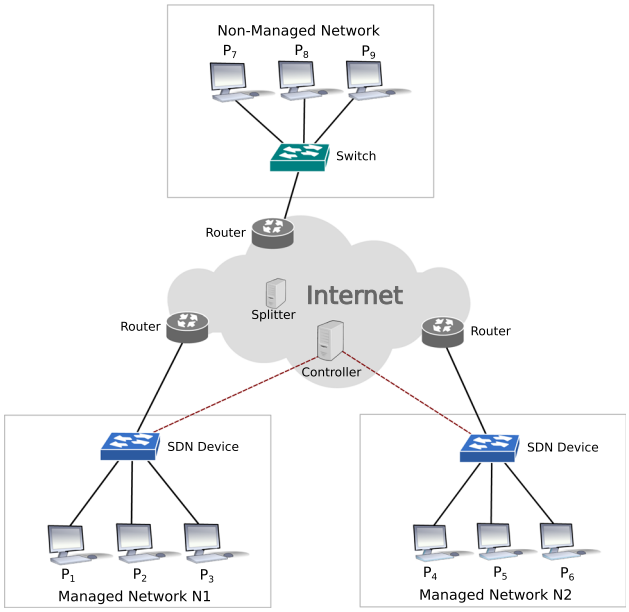

Fig. 3 An example of a network architecture with SDN and non-SDN devices running the P2PSP with our SDN approach

– SDN devices are the gateways of a set of peers (see Fig. 3). In this case, the scrambling can be performed in the outgoing chunks, but also in the incoming chunks if the destination EP is also connected to a SDN device. Since the controller is a centralized entity, it takes control of the scrambling by analyzing the new packets and sending the appropriate flows to the respective devices.

In order to achieve the goal of applying the studied solution to mixed environments, we have added a minimal update to the P2PSP Ryu app. It consists of adding a list of devices managed by the controller and informs on which peers are connected to each managed device. With this modification, we can verify whether a message has been previously modified by the controller or not. The idea is to avoid a double EP scrambling and to make sure that all peers receive all chunks once.

For the sake of simplicity, let us assume that there is a team with $Z = \{Z_{\text{in}} \cup Z_{\text{out}}\}$ peers. One of them is an MP and another is a TP. The managed network has Z_{in} peers, including the TP. So, Z_{out} peers are outside the managed network. We can differentiate four possible attacking scenarios:

1. Attacks happen outside the managed network:
When an attack occurs between peers in Z_{out}, the attack cannot be detected because the scrambling algorithm is not working among these peers and there is no outside TP. However, this attack does not affect peers in Z_{in}.

2. Attacks happen inside the managed network:
   Attacks among peers in Z_{in} will be detected thanks to the scrambling approach, using $Z = Z_{in}$ in (2) or (3).
3. Attacks from outside to inside the managed network:
   If a peer in Z_{in} is attacked by the MP in Z_{out}, the attack is detected because the scrambling can change the destination peer to another peer in the managed network that could be the TP. In this case, the probability of detection is given by using $Z = Z_{in} + 1$ in (2) or (3), i.e., the number of peers inside the managed network plus the MP. Equations are correct only if the MP attacks the managed network in all rounds.
4. Attacks from inside to outside a managed network:
   When the MP in Z_{in} attacks a peer in Z_{out}, the attack is not detected. Although the attack does not cause damage in the SDN environment, it affects peer in Z_{out}. To avoid this, we could improve the proposed algorithm by using two different scrambling tables:

   (a) A list with peers in Z_{in}, which is used for scrambling when the chunks come from outside.
   (b) A list with all peers, which is used for scrambling when the chunks leave the managed network.

   The pseudocode for the Ryu controller including this improvement is shown in Algorithm 2.
   Therefore, the probability of detection (2) or (3) is reduced by $\frac{Z_{in}}{Z}$, which is the probability of redirecting the MP's message to the managed network.

Previous considerations can be extended to instances with several managed networks where SDN devices are controlled by the same controller and the scrambling table contains the Z_{in} peers in all managed networks.

Consequently, the studied solution works in a mixed network maintaining some of the benefits as in a fully SDN architecture.

Algorithm 2 P2PSP Ryu app algorithm for mixed networks

INPUT: peer_list_b, packet, splitter

```
peer_list_a = remove_external_peers(peer_list_b)   #Remove peers outside the management networks
scrambling_table_outside=dict(zip(peer_list_a, peer_list_a)) #Create a scrambling table associating original EPs with new destinations EPs for using among peers outside (only includes peers into the management networks)
scrambling_table_inside=dict(zip(peer_list_b, peer_list_b))  #Create a dictionary for associating original EPs with new destinations EPs for using among peers inside (it includes all peers into the team)

if P2PSP UDP packet then
    if splitter's packet and new round then
        clean_flows()                          #Remove rules from devices
        shuffle(scrambling_table_outside)      #Assign a new EP mapping
        shuffle(scrambling_table_inside) #Assign a new EP mapping
    else
        if packet comes from outside then
            scrambling_table = scrambling_table_outside
        else
            scrambling_table = scrambling_table_inside
        if packet destination in scrambling_table then
            store mapping for IP to Mac and Mac to Port
            if scrambling_table[destination] is myself then
                destination = myself
            new_destination = scrambling_table[destination]
            add_Flow(new_destination)          #Add new rules for the devices
send_PacketOut()                               #Send modified packet
```

5 Experiments

In all the simulations shown in this section, peers are in a managed network. Simulations were carried out with Mininet network simulation tool [23], on an Intel Core i7 CPU 860 @ 2.80GHz x 8 machine with 16 GB of RAM, using Ryu SDN Framework as the SDN Controller [17].

5.1 Fixed versus variable targets attacks

Assuming EPs are shuffled in each round, an MP could think that changing the target could reduce the probabilities of being detected, especially if the scrambling is pseudo-random, meaning a totally different EP association is forced for each round. In that case, using a variable target could prove helpful because, if the target is fixed after n rounds, it means the MP has attacked the entire team. However, the same does not occur in the case of a totally random scrambling because the EP mapping is unpredictable. For this reason, we decided to implement a totally random mapping. In this scenario, the probabilities of an MP being detected should be the same regardless of the attack method, whether it be fixed or variable.

We run a set of experiments with the following parameters:

- *Team size* from 10 to 100.
- *Samples* 100 samples (executions) for each instance.
- *Period* 1, i.e., all the executions perform one scrambling per round.
- *Cases* fixed and variable target attacks. One per round.

Results of the experiments are shown in Fig. 4. For both fixed target and variable target attacks, the results fit quite well with the probability model p in (2). An MP is usually detected before 60 percent of the number of rounds equals the team size. This percentage is the same as the average percentage of vulnerable computers contacted as the normalized shuffle rate increases, which was calculated in [24]. Whereas the perfect shuffling cost results in a drawback due to the rel-

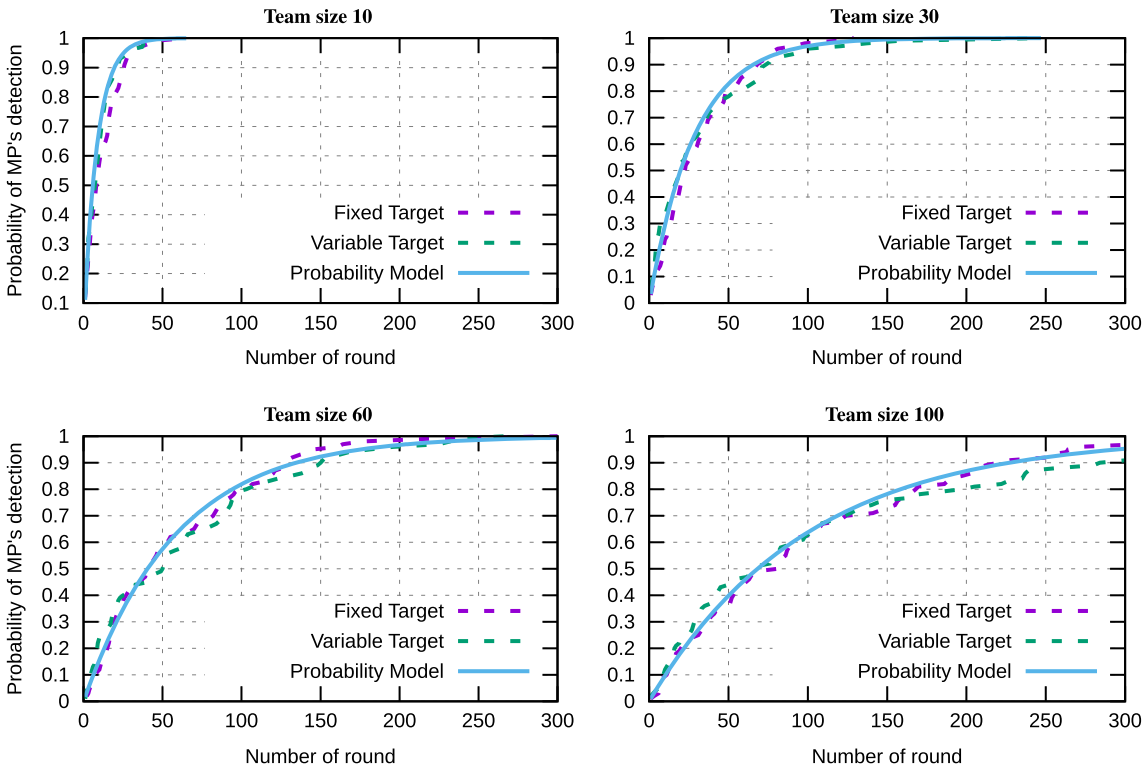

Fig. 4 Experimental detection probabilities of an MP performing one attack per round. Scrambling period = 1

atively high percentage for an attacker to find a vulnerable host, in our case the scrambling is worth its cost due to the relatively high percentage to reach the trusted peer.

This justifies the idea of keeping the EP mapping totally random in order to avoid a smart attack where MPs can figure out how the dynamic scrambling is carried out for a hypothetical pseudo-random system. For all team size configurations, the probability of detection is very close to 1 when the number of rounds triples the size of the team. For instance, in a team with 100 peers, the probability of detection before round 300 is 0.95. Thus, assuming an average bitrate of 2500 kbps (typical HD 720p video), and a chunk size of 1024 bytes, the MP will be detected and expelled before $t = 98.3$ s.

Regarding the consequences for the detection system when opting for a fixed or variable type of attack, we conclude that the probabilities of detection are quite similar. Therefore, an MP will be detected and expelled in the same way regardless of the type of attack it carries out. Additionally, a variable attack makes no sense in a traditional network without scrambling because after each target change is made, the malicious peer is nearer to being discovered. Moreover, MPs performing more than one attack per round will increase their probability of being detected.

5.2 Modifying the scrambling period

Modifying the scrambling period parameter helps us to reduce the CPU usage, network traffic and the number of flows installed in devices. However, it causes a loss in the efficiency of MP detection. Figure 5 shows a set of experiments carried out for a team of 100 peers varying the scrambling period (100 samples for each one).

The simulated MP detection is close to the probability model shown in Eq. (3). As expected, the number of average rounds to detect the MP with a probability greater than 0.9 increases significantly with the scrambling period.

6 Conclusions and future work

In this study, we analyze a moving target defense based on scrambling the target EPs in SDN devices in order to detect malicious peers in peer-to-peer fully connected push-based overlays by using trusted peers. The experiments demonstrate the viability of our proposal, resulting in a relatively quick MP detection (in relation to the overlay size), not only in pure SDN environments but also in mixed environments where some peers are on the Internet and others are under managed networks.

Since modifying the scrambling period has a severe impact on the performance of the detection algorithm, we propose

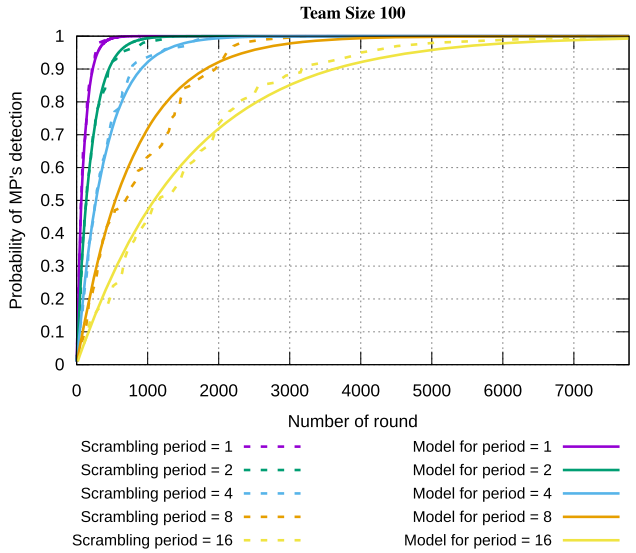

Fig. 5 Experimental results for MP detection probabilities with different scrambling periods

optimizations such as a proactive approach and controller location techniques as possible future lines of work.

Acknowledgements The authors gratefully thank the referees for the constructive comments and recommendations which certainly helped to improve the quality of the paper.

Funding This paper has been supported by the Spanish Ministry (RTI2018-095993-B-100), in part financed by the European Regional Development Fund (ERDF). This publication has emanated from research supported in part by a research Grant from Science Foundation Ireland (SFI) under Grant Numbers 13/SIRG/2178 and 16/RC/3918. Cristóbal Medina-López is supported by an FPU Fellowship (FPU14/00635) from the Spanish Ministry of Education (MECD).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This paper does not contain any studies with human participants or animals performed by any of the authors.

References

1. Selcuk, A.A., Uzun, E., Pariente, M.R.: A reputation-based trust management system for p2p networks. In: IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004, pp. 251–258. IEEE (2004)
2. Marti, S., Garcia-Molina, H.: Limited reputation sharing in p2p systems. In: Proceedings of the 5th ACM Conference on Electronic Commerce, pp. 91–101. ACM (2004)
3. Su, X., Dhaliwal, S.K.: Incentive mechanisms in p2p media streaming systems. IEEE Internet Comput. **14**(5), 74–81 (2010)
4. Wang, E.K., Li, Y., Ye, Y., Yiu, S.M., Hui, L.C.K.: A dynamic trust framework for opportunistic mobile social networks. IEEE Trans. Netw. Serv. Manag. **15**(1), 319–329 (2018)
5. Medina-López, C., González-Ruiz, V., Casado, L.G.: On mitigating pollution and free-riding attacks by shamir's secret sharing in fully connected p2p systems. In: 2017 13th International on Wireless Communications and Mobile Computing Conference (IWCMC), pp. 711–716. IEEE (2017)
6. Medina-López, C., Shakirov, I., Casado, L.G., González-Ruiz, V.: On pollution attacks in fully connected P2P networks using trusted peers. In: Intelligent Systems Design and Applications, pp. 144–153, Springer, Cham, Porto (2017)
7. P2PSP Team. Peer to peer straightforward protocol. https://p2psp.org/. Accessed 28 June 2019
8. Yang, S., Jin, H., Li, B., Liao, X.: A modeling framework of content pollution in peer-to-peer video streaming systems. Comput. Netw. **53**(15), 2703–2715 (2009)
9. Markowitch, O., Roggeman, Y.: Probabilistic non-repudiation without trusted third party. Second Conf. Secur. Commun. Netw. **99**, 25–36 (1999)
10. Wang, C., Lu, Z.: Cyber deception: overview and the road ahead. IEEE Secur. Priv. **16**(2), 80–85 (2018)
11. Albanese, M., Huang, D.: MTD 2018: 5th ACM workshop on moving target defense (MTD). In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pp. 2175–2176. ACM (2018)
12. Zheng, J., Namin, A.S.: A survey on the moving target defense strategies: an architectural perspective. J. Comput. Sci. Technol. **34**(1), 207–233 (2019)
13. Zhou, X., Lu, Y., Wang, Y., Yan, X.: Overview on moving target network defense. In: 2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC), pp. 821–827 (2018)
14. Antonatos, S., Akritidis, P., Markatos, E.P., Anagnostakis, K.G.: Defending against hitlist worms using network address space randomization. Comput. Netw. **51**(12), 3471–3490 (2007)
15. Jafarian, J.H., Al-Shaer, E., Duan, Q.: Openflow random host mutation: transparent moving target defense using software defined networking. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, pp. 127–132. ACM (2012)
16. Cai, G., Wang, B., Wang, X., Yuan, Y., Li, S.: An introduction to network address shuffling. In: 2016 18th International Conference on Advanced Communication Technology (ICACT), pp. 185–190. IEEE (2016)
17. Ryu SDN Framework. https://osrg.github.io/ryu/. Accessed 28 June 2019
18. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
19. Green, M., MacFarland, D.C., Smestad, D.R., Shue, C.A.: Characterizing network-based moving target defenses. In: Proceedings of the Second ACM Workshop on Moving Target Defense, pp. 31–35. ACM (2015)
20. MacFarland, D.C., Shue, C.A.: The sdn shuffle: creating a moving-target defense using host-based software-defined networking. In: Proceedings of the Second ACM Workshop on Moving Target Defense, pp. 37–41. ACM (2015)
21. Pfaff, B., Lantz, B., Heller, B., Barker, C., Beckmann, C., Cohn, D., Talayco, D., Erickson, D., McDysan, D., Ward, D., et al.: Openflow switch specification v1.3.1 (2012)
22. Killi, B.P.R., Rao, S.V.: Capacitated next controller placement in software defined networks. IEEE Trans. Netw. Serv. Manag. **14**(3), 514–527 (2017)
23. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the

9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, pp. 19:1–19:6. ACM, New York (2010)
24. Carroll, T.E., Crouse, M., Fulp, E.W., Berenhaut, K.S.: Analysis of network address shuffling as a moving target defense. In: 2014 IEEE International Conference on Communications (ICC), pp. 701–706. IEEE (2014)