Preventing Distributed Denial-of-Service Flooding Attacks with Dynamic Path Identifiers

Hongbin Luo, *Member, IEEE,* Zhe Chen, Jiawei Li, and Athanasios V. Vasilakos, *Senior Member, IEEE*

***Abstract*—In recent years, there are increasing interests in using path identifiers ($PIDs$) as inter-domain routing objects. However, the $PIDs$ used in existing approaches are static, which makes it easy for attackers to launch distributed denial-of-service (DDoS) flooding attacks. To address this issue, in this paper, we present the design, implementation, and evaluation of D-PID, a framework that uses $PIDs$ negotiated between neighboring domains as inter-domain routing objects. In D-PID, the PID of an inter-domain path connecting two domains is kept secret and changes dynamically. We describe in detail how neighboring domains negotiate $PIDs$, how to maintain ongoing communications when $PIDs$ change. We build a 42-node prototype comprised by six domains to verify D-PID's feasibility and conduct extensive simulations to evaluate its effectiveness and cost. The results from both simulations and experiments show that D-PID can effectively prevent DDoS attacks.**

***Index Terms*—Inter-domain routing, security, distributed denial-of-service (DDoS) attacks, path identifiers.**

I. Introduction

DISTRIBUTED denial-of-service (DDoS) flooding attacks are very harmful to the Internet. In a DDoS attack, the attacker uses widely distributed zombies to send a large amount of traffic to the target system, thus preventing legitimate users from accessing to network resources [1]. For example, a DDoS attack against BBC sites in Jan. 2016 reached 602 gigabits per second and "took them down for at least three hours" [3]. More recently, the hosting provider OVH suffered a large scale DDoS attack in Sep. 2016, launched by a botnet composed at least of 150,000 Internet-of-things (IoT) devices. This attack peaked at nearly one terabit per second (Tbps) and even forced Akamai to stop offering DDoS protection to OVH [2]. Therefore, many approaches [4] have been proposed in order to prevent DDoS flooding attacks, including network ingress filtering [5] - [9], IP traceback [10] - [14], capability-based designs [15] - [18], and shut-up messages [19] - [20].

Manuscript received July 22, 2016, revised Dec. 28, 2016, accepted March 23, 2017. This work was supported in part by the Natural Science Foundation of China (NSFC) under Grant Nos. 61422101, 61271200, and 61232017, in part by the "973 Program" of China under Grant No. 2013CB329106, and in part by the Fundamental Research Funds for the Central Universities under Grant No. 2016JBZ002.

H. Luo is with the School of Computer Science and Engineering, Beihang University, Beijing 100083, China. This work was done when he was in Beijing Jiaotong University, Beijing 100044, China (Email: hbluo@bjtu.edu.cn).

Z. Chen is now with Huawei Technologies. This work was done when he was in Beijing Jiaotong University, Beijing 100044, China (Email: chenzhe17@huawei.com).

J. Li is with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (Email: 15111049@bjtu.edu.cn).

A. V. Vasilakos is with the Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology, 97187 Lulea, Sweden (Email: athanasios.vasilakos@ltu.se).

Digital Object Identifier:

At the same time, in recent years there are increasing interests in using path identifiers $PIDs$ that identify paths between network entities as inter-domain routing objects, since doing this not only helps addressing the routing scalability and multi-path routing issues [21], but also can facilitate the innovation and adoption of different routing architectures [22]. For instance, Godfrey *et al.* proposed pathlet routing [21], in which networks advertise the $PIDs$ of pathlets throughout the Internet and a sender in the network constructs its selected pathlets into an end-to-end source route. Koponen *et al.* further argued in their insightful architectural paper that using pathlets for inter-domain routing can allow networks to deploy different routing architectures, thus encouraging the innovation and adoption of novel routing architectures [22]. Jokela *et al.* proposed in LIPSIN [23] to assign identifiers to links in a network and to encode the link identifiers along the path from a content provider to a content consumer into a zFilter (*i.e.*, a PID), which is then encapsulated into the packet header and used by routers to forward packets. Luo *et al.* proposed an information-centric internet architecture called CoLoR [24] that also uses $PIDs$ as inter-domain routing objects in order to enable the innovation and adoption of new routing architectures, as in [22].

There are two different use cases of $PIDs$ in the aforementioned approaches. In the first case, the $PIDs$ are globally advertised (as in pathlet routing [21] and [22]). As a result, an end user knows the $PID(s)$ toward any node in the network. Accordingly, attackers can launch DDoS flooding attacks as they do in the current Internet. In the second case, conversely, $PIDs$ are only known by the network and are secret to end users (as in LIPSIN [23] and CoLoR [24]). In the latter case, the network adopts an information-centric approach [25] - [27] where an end user (*i.e.*, a content provider) knows the $PID(s)$ toward a destination (*i.e.*, a content consumer) only when the destination sends a content request message to the end user. After knowing the $PID(s)$, the end user sends packets of the content to the destination by encapsulating the $PID(s)$ into the packet headers. Routers in the network then forward the packets to the destination based on the $PIDs$.

It seems that keeping $PIDs$ secret to end users (as in [23], [24]) makes it difficult for attackers to launch DDoS flooding attacks since they do not know the $PIDs$ in the network. However, keeping $PIDs$ secret to end users is not enough for preventing DDoS flooding attacks if $PIDs$ are static. For example, Antikainen *et al.* argued that an adversary can construct novel zFilters (*i.e.*, $PIDs$) based on existing ones and even obtain the link identifiers through reverse-engineering, thus launching DDoS flooding attacks [28]. Moreover, as it is shown in Sec. II-B, attackers can launch DDoS flooding attacks by learning $PIDs$ if they are static.

To address this issue, in this paper, we present the design, implementation and evaluation of a dynamic PID (D-PID) mechanism. In D-PID, two adjacent domains periodically update the $PIDs$ between them and install the new $PIDs$ into the data plane for packet forwarding. Even if the attacker obtains the $PIDs$ to its target and sends the malicious packets successfully, these $PIDs$ will become invalid after a certain period and the subsequent attacking packets will be discarded by the network. Moreover, if the attacker tries to obtain the new $PIDs$ and keep a DDoS flooding attack going, it not only significantly increases the attacking cost (Sec. V-A1), but also makes it easy to detect the attacker (Sec. V-A2). In particular, our main contributions are two fold.

On one hand, we propose the D-PID design by addressing the following challenges. First, how and how often should $PIDs$ change while respecting local policies of autonomous systems ($ASes$)? To address this challenge, D-PID lets neighboring domains negotiate the $PIDs$ for their inter-domain paths based on their local policies (Sec. III-B). In particular, two neighboring domains negotiate a PID-prefix (as an IP-prefix) and a PID update period for every inter-domain path connecting them. At the end of a PID update period for an inter-domain path, the two domains negotiate a different PID (among the PID-prefix assigned to the path) to be used in the next PID update period. In addition, the new PID of an inter-domain path is still kept secret by the two neighboring domains connected by the path.

Second, since inter-domain packet forwarding is based on $PIDs$ that change dynamically, it is necessary to maintain legitimate communications while preventing illegal communications when the $PIDs$ change. To address this challenge, D-PID lets every domain distribute its $PIDs$ to the routers in the domain (Sec. III-C). For every inter-domain path, the routers in a domain forward data packets based on the PID of the previous PID update period and that of the current PID update period. In addition, D-PID uses a mechanism similar to the one that the current Internet collects the minimum MTU (maximum transmission unit) of networks so that a content consumer knows the minimum update period of $PIDs$ along the path from a content provider to it (Sec. III-D - Sec. III-F). Based on this period, the content consumer periodically resends a content request message to the network in order to renew the $PIDs$ along the path.

Third, the overheads incurred by changing $PIDs$ should be kept as small as possible. This includes not only the overhead in negotiating $PIDs$ by neighboring domains, but also the overhead for a domain to distribute the updated $PIDs$ to routers in the domain, and that for transmitting content request messages resent by content consumers. To address this challenge, the PID prefix assigned to an inter-domain path is unique among the PID prefixes assigned by the two domains connected by the inter-domain path.

On the other hand, we build a 42-node prototype (Sec. IV) comprised by six domains to verify D-PID's feasibility and conduct extensive simulations (Sec. V) to evaluate D-PID's effectiveness and overheads. Our results show that D-PID does help preventing DDoS flooding attacks since it not only imposes significant overhead for the attacker to launch DDoS flooding attacks, but also makes it easier for the network to detect the attacker. Surprisingly, achieving such benefits only incurs little overheads. Our simulation results show that the number of extra content request messages caused by D-PID is only 1.4% or 2.2% (by using different data traces), when the PID update period is 300 seconds. Even if the PID update period is 30 seconds, the peak PID update rate of a domain is less than 10 per second with a probability higher than 95%, and the maximal PID update rate of all domains is 202 per second, which is significantly less than the peak update rate (1,962 per second) of IP-prefixes in the current Internet [32].

While part of this work has been published in [45], we significantly extend it with the following new contributions. First, we propose an approach for neighboring domains to negotiate $PIDs$ (Sec. III-B) and to distribute them to routers in a domain (Sec. III-C). Second, we implemented D-PID in a prototype to verify its feasibility (Sec. IV). Third, we conduct extensive simulations to evaluate the effectiveness of D-PID in defending against DDoS flooding attacks (Sec. V-A).

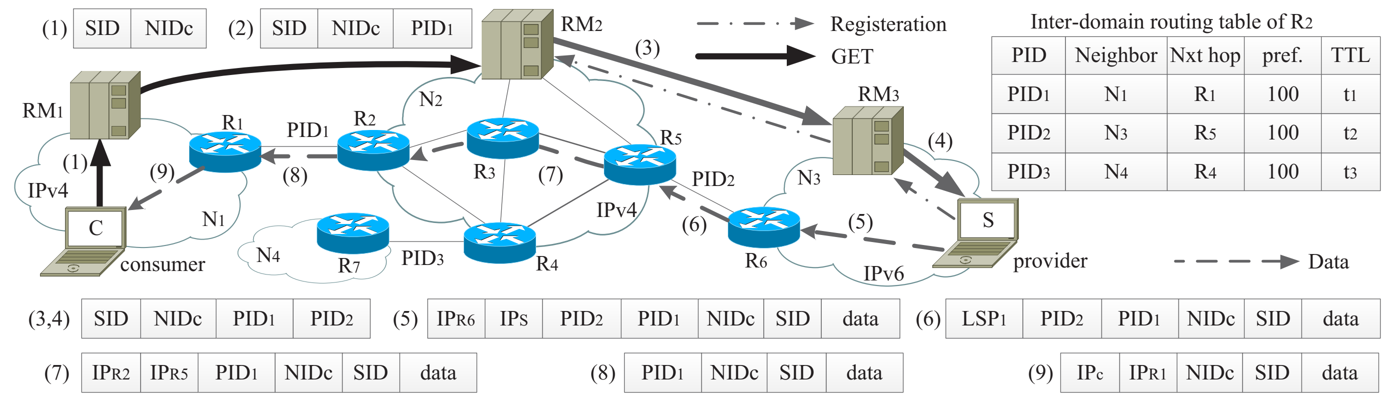
The rest of this paper is organized as follows. In Section II, we briefly introduce CoLoR and present two approaches for learning $PIDs$ when $PIDs$ are static. In Section III, we describe the design details of D-PID. In Section IV, we show the implementation of D-PID. In Section V, we present the results from extensive simulations. In Section VI, we outline related work and compare D-PID with them. Finally, we conclude the paper in Section VII.

II. Background and Motivation

In this section, we first make a brief introduction to CoLoR because it lays the foundation of our study in this paper. We then describe why we should dynamically change $PIDs$ in II-B, thus motivating our study. Note that D-PID can be applied to other Internet architectures that use secret $PIDs$ as inter-domain routing objects (such as LIPSIN [23]).

A. Brief Introduction to CoLoR

CoLoR is a receiver-driven information centric network architecture that assigns unique and persistent content names (or service identifiers, $SIDs$) to content chunks. As in [20] and [27], CoLoR assigns intrinsic secure self-certifying node identifiers ($NIDs$) to network nodes and ASes so that authenticating a node/AS does not require an external authority such as ICANN, thus improving security and privacy. In addition, two neighboring domains negotiate a PID for every inter-domain path between them and the PID is only known by them. The two domains then use the $PIDs$ assigned to their inter-domain paths to forward packets from one domain to the other. For this purpose, the routers in a domain maintains an inter-domain routing table, which records the PID of each inter-domain path and the border router that the PID originates, as illustrated at the upper right corner in Fig. 1. For instance, the border router in domain N_2 connecting PID_2 in Fig. 1 is R_5. On the other hand, each domain is free to choose its preferred intra-domain routing architecture so that a domain



Inter-domain routing table of R2

PID	Neighbor	Nxt hop	pref.	TTL
PID1	N1	R1	100	t1
PID2	N3	R5	100	t2
PID3	N4	R4	100	t3

Fig. 1. Illustration for the basic operations in CoLoR.

A uses IPv4 for intra-domain routing while another domain B may use IPv6 for intra-domain routing.

Furthermore, every domain in the Internet maintains a logically centralized (but may be physically distributed) resource manager (RM) used to propagate the reachability information of $SIDs$. Particularly, when a content provider wants to provide a content chunk to consumers, he registers the SID of the content chunk to its local RM. The local RM then registers the SID to its providers or peers, by using an approach similar to the one used in [26].

When a content consumer wants to obtain a piece of content, it sends out a GET message to its local RM. If the desired content is hosted by a local node, the RM forwards the GET message to that node. Otherwise, the RM forwards the GET message to the RM in a neighboring domain (toward the content provider) over a secure channel between the two RMs (because of the use of intrinsic secure identifiers). During this process, the $PIDs$ of inter-domain paths from the content provider to the content consumer are determined. The content provider then sends the desired content to the content consumer by embedding the collected $PIDs$ into headers of packets for the desired content.

Fig. 1 illustrates the basic content retrieval process in CoLoR, assuming that the content provider holds a piece of desired content with name SID and the content consumer wants to obtain the content. Firstly, the content provider registers the SID to its local RM (*i.e.*, RM_3), which then registers the SID to its provider RM (*i.e.*, RM_2), as illustrated by the dashed lines in Fig. 1. When the content consumer wants to obtain the content, it sends a GET message to its local RM (*i.e.*, RM_1), as illustrated by (1) in Fig. 1. Since no local host can provide the desired content, RM_1 appends PID_1 at the end of the GET message and forwards the GET message to its neighbor RM_2, as illustrated by (2) in Fig. 1. Similarly, RM_2 appends PID_2 at the end of the GET message and forwards the GET message to RM_3, as illustrated by (3) in Fig. 1. RM_3 then forwards the GET message to the content provider who holds the desired content, as illustrated by (4) in Fig. 1.

When the content provider receives the GET message, it encapsulates the obtained $PIDs$ into headers of packets for the desired content, and sends the data packets to the content consumer. When the data packets enter a domain, the ingress border router encapsulates them an outer header corresponding to the routing protocol used by the domain. By contrast, when a data packet leaves a domain, the egress border router removes the outer header. For instance, when router R_5 receives a data packet carrying PID_1, it encapsulates the data packet with an IPv4 header (as illustrated by (7) in Fig. 1), if domain N_2 uses IPv4 for intra-domain routing. When router R_2 receives the data packet, it removes the outer IPv4 header, as illustrated by (8) in Fig. 1. Note that the outermost PID is popped out by the ingress border router of each domain in order to prevent content consumers from knowing the $PIDs$ toward a content provider and launching DDoS attacks. For example, when the border router R_5 receives the data packet, it removes PID_2 since it is the outermost PID, as can be seen by comparing (6) and (7) in Fig. 1.

CoLoR offers several interesting features. First, as an information-centric network architecture, routers in the network can locally cache the popular contents so as to serve nearby users, thus reducing redundant transmission and content retrieval delay. To achieve this, the router caching a piece of content simply needs to register the SID of the content to its local RM. Second, it is easy to accurately, timely estimate the traffic matrices of a network since an ingress border router of a domain can know the egress border router of a packet by looking up the inter-domain routing table [29]. Third, CoLoR makes it easy to efficiently integrate information-centric networking and software-defined networking [30]. In addition, the data plane in CoLoR is scalable because the sizes of inter-domain routing tables maintained by border routers depend on the number of neighbors of a domain, which is limited to be several thousands in the Internet today. While RMs needs to deal with $SIDs$ whose number is quite large, the RM in a large domain can be realized by using a distributed system (*e.g.*, a data center) and the content names could be aggregated by using appropriate name formats such as $P : L$ [26]. Furthermore, to upgrade from the current Internet to CoLoR, the intra-domain routers of domains do not need to be upgraded, thus reducing deployment cost. Finally, CoLoR offers some security benefits [31] while avoiding Interest flooding attacks suffered by NDN [44] because 1) both routers and RMs in CoLoR do not maintain pending Interest tables; and 2) the $PIDs$ carried in GET messages can be used to trace

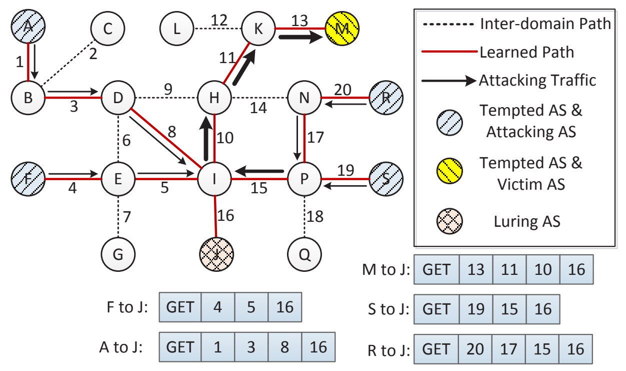


Fig. 2. Illustration for the GET luring.

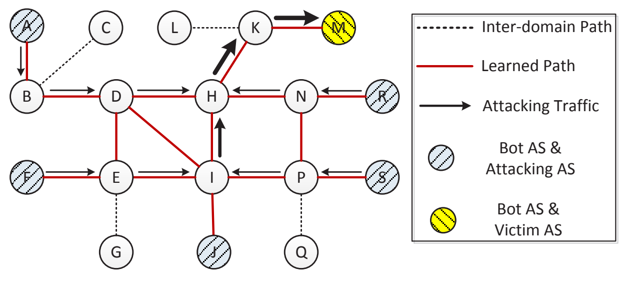


Fig. 3. Illustration for the botnet cooperation.

back attackers.

CoLoR also has some drawbacks that need to be addressed before its real deployment in the future. First, carrying the NID of the content consumer and the desired SID in packet headers reveals user privacy. Second, border routers need to encapsulate/decapsulate outer packet headers (*e.g.*, IPv4 headers), which makes it challenging to realize line-speed packet forwarding. Third, as it is shown below, attackers can learn $PIDs$ in the network and launch DDoS attacks in the data plane, if $PIDs$ are static. As an attempt to address these drawbacks, in this paper we propose D-PID to prevent DDoS attacks in the data plane.

B. *Why Dynamically Changing PIDs*

In this subsection, we explain why it is necessary to dynamically change $PIDs$ in CoLoR. To this end, we first present two approaches to learning $PIDs$ whey they are static. We then present an example to show that an attacker can launch DDoS attacks when he have learnt some $PIDs$ in the network.

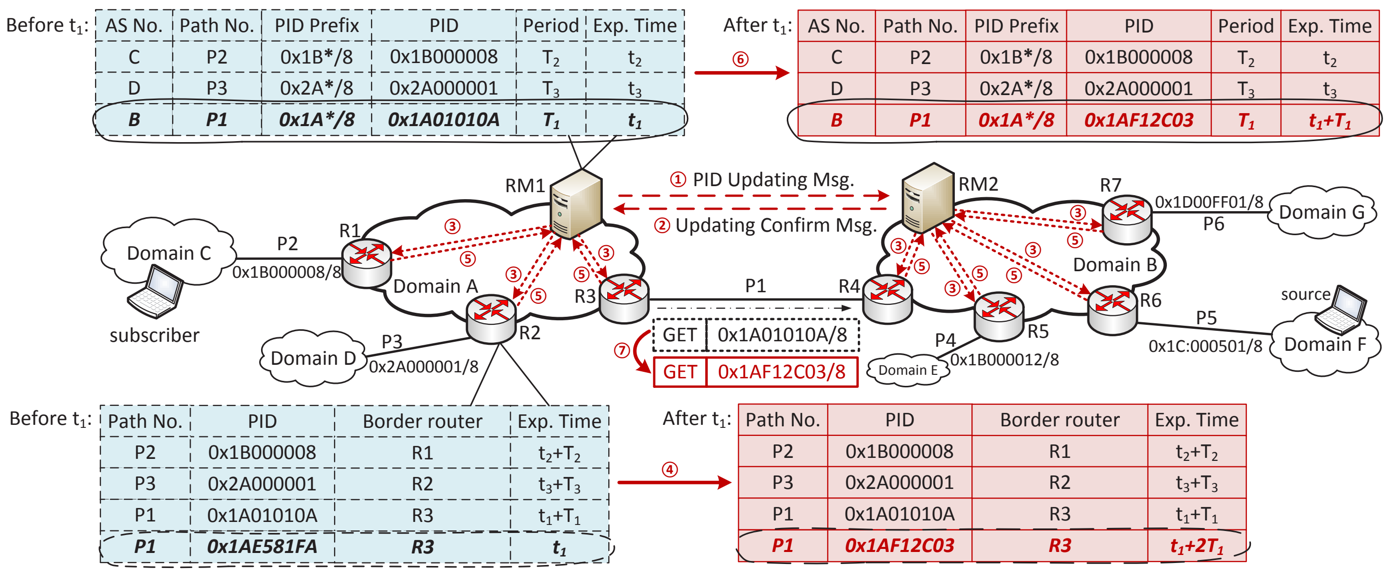
1) Two approaches to learning PIDs: The first approach to learning $PIDs$ is ***GET Luring***, where an attacker uses an end host to register normal content names into the network, thus luring GET messages from content consumers. Since the corresponding $PIDs$ are carried by the GET messages, the attacker then can learn a part of $PIDs$ in the network. We call such a process as the PID learning stage in the rest of this paper. Fig. 2 illustrates the process of GET luring. For ease of presentation, we call the AS where the attacker locates as a luring AS and the $ASes$ that send GET messages to the luring AS as tempted $ASes$. Each node in Fig. 2 represents an AS in the Internet, AS J is the luring AS, and $ASes$ A, F, M, R, and S are the tempted $ASes$. At the beginning, AS J registers content names into the network. Then, $ASes$ A, F, M, R, and S are lured to send GET messages to AS J. The GET messages received by AS J are shown at the bottom of Fig. 2. The attacker then learns the corresponding $PIDs$ in the network, which are represented by solid lines in Fig. 2.

Another approach to learning $PIDs$ is ***botnet cooperation***. In botnet cooperation, an attacker is assumed to have controlled a distributed botnet by using various methods such as worms or instant messaging applications. In particular, zombies in the botnet register content names to the network and send GET messages mutually, thus learning the $PIDs$ in the network. Fig. 3 illustrates botnet cooperation, assuming that an AS in which one or more bots locate is called a bot AS. In Fig. 3, $ASes$ A, F, M, R, and S are bot $ASes$. When a zombie in bot AS A sends a GET message to another zombie in bot AS R, the inter-domain paths $A-B$, $B-D$, $D-H$, $H-N$, and $N-R$ are learned. Similarly, other inter-domain paths marked with bold lines in Fig. 3 can also be learned.

2) Launching DDoS Attacks: Once the attacker has learned a part of $PIDs$ in the network (through GET luring, botnet cooperation, or other possible approaches), it can freely send packets along the paths represented by the learned $PIDs$. We assume that the attacker can compromise a number of computers along the paths as zombies, by using similar methods with the ones in the current Internet (*e.g.*, by using worms). Note that this is a pessimistic assumption since the integrality of a content in information-centric networking is usually easy to verify [25] - [27]. Then the attacker can order the zombies to flood a victim that should also be along the learned paths. We call such a process as the attacking stage.

Fig. 2 and Fig. 3 illustrate the attacking stage. We call the $ASes$ where the compromised computers (that flood the victim) locate as attacking $ASes$ and the AS where the victim locates as the victim AS. Note that an AS may play multiple roles, *e.g.*, a tempted AS at the PID learning stage may be an attacking AS at the attacking stage. In Fig. 2, AS M is the victim AS, and $ASes$ A, F, R, and S are the attacking $ASes$ that are compromised by the attacker and can flood the victim by using the learned $PIDs$, as illustrated by the arrowed lines in Fig. 2. In Fig. 3, AS M is the victim AS, $ASes$ A, F, J, R, and S are the attacking $ASes$, and the attacking traffic is represented by the arrowed lines.

From the above descriptions, one can see that it is possible for an attacker to launch DDoS attacks if $PIDs$ are kept secret but static. In addition, since the $PIDs$ carried by data packets are popped out domain-by-domain, the victim does not know the $PIDs$ to the attackers. Accordingly, it cannot trace back them. One may argue that we should not pop out the $PIDs$ when data packets pass through domains. In that case, however, an attacker can try to hide himself by prepending some invalid $PIDs$ at data packets. For instance, the actual $PIDs$ from the content provider S to the content consumer C in Fig. 1 are PID_2 and PID_1. In order to hide himself, S can prepend an invalid PID (*e.g.*, PID_6 not shown in Fig. 1) before PID_2 and PID_1. This way, the content consumer C cannot easily find S even if we do not pop out $PIDs$ during the



Before t_1:

AS No.	Path No.	PID Prefix	PID	Period	Exp. Time
C	P2	0x1B*/8	0x1B000008	T_2	t_2
D	P3	0x2A*/8	0x2A000001	T_3	t_3
B	***P1***	***0x1A*/8***	***0x1A01010A***	T_1	t_1

After t_1:

AS No.	Path No.	PID Prefix	PID	Period	Exp. Time
C	P2	0x1B*/8	0x1B000008	T_2	t_2
D	P3	0x2A*/8	0x2A000001	T_3	t_3
B	***P1***	***0x1A*/8***	***0x1AF12C03***	T_1	t_1+T_1

Before t_1:

Path No.	PID	Border router	Exp. Time
P2	0x1B000008	R1	t_2+T_2
P3	0x2A000001	R2	t_3+T_3
P1	0x1A01010A	R3	t_1+T_1
P1	***0x1AE581FA***	***R3***	t_1

After t_1:

Path No.	PID	Border router	Exp. Time
P2	0x1B000008	R1	t_2+T_2
P3	0x2A000001	R2	t_3+T_3
P1	0x1A01010A	R3	t_1+T_1
P1	***0x1AF12C03***	***R3***	t_1+2T_1

Fig. 4. Illustration for PID negotiation and PID update in D-PID.

packet forwarding process. Therefore, we propose to defend against DDoS attacks by dynamically changing $PIDs$.

III. The D-PID Design

In this section, we first present an overview of D-PID, and then describe the design details.

A. Overview of D-PID

From Sec. II-B, one can see that an attacker can learn a part of the $PIDs$ used by domains in the Internet and launch attacks, if the $PIDs$ are static. Thus, the core idea of D-PID is to dynamically change the PID of an inter-domain path. In particular, for a given (virtual) path connecting two neighboring domains A and B, it is assigned a PID and an update period T_{PID}. The update period T_{PID} represents how long the PID of the path should be changed since the PID is assigned. For instance, if path P_1 in Fig. 4 is assigned PID_1 at time t, the RMs in the two domains should negotiate a new PID (*i.e.*, PID_2) for P_1 at time $t + T_{PID}$ and a new update period T'_{PID}, by using the negotiation process described in Sec. III-B. At time $t + T_{PID} + T'_{PID}$, the two RMs will negotiate another new PID (*i.e.*, PID_3) for P_1.

Once the new PID (*i.e.*, PID_2) is assigned to the path, the RMs in domains A and B then distribute the new PID (*i.e.*, PID_2) to the routers in domains A and B (Sec. III-C). After that, the RMs append the new PID (*i.e.*, PID_2) onto GET messages if the path is chosen to carry the corresponding data packets. At the same time, the border routers forward data packets based on the new PID (*i.e.*, PID_2). Since some GET packets are forwarded from domain A (or B) to domain B (or A) by using the old PID (*i.e.*, PID_1) of the path, the old PID is still valid until $t + T_{PID} + T'_{PID}$. Without loss of generality, we assume that T_{PID} equals to T'_{PID} in the rest of this paper. That is, the update period of a path is fixed.

Note that the new PID of the path is still known only by the two domains. However, it is possible that a communication lasts longer than two update periods. Thus, when the PID of the path changes to PID_3, ongoing communications may be interrupted. To address this issue, in Sec. III-F we propose a mechanism similar to the one that the current Internet collects the minimum MTU of networks so that a content consumer knows the minimum update period of $PIDs$ along the path from a content provider to it. Based on this period, the content consumer then re-sends a GET message to the network in order to renew the $PIDs$ along the path.

Note also that in D-PID, all domains should dynamically change the $PIDs$ of its inter-domain paths. Depending on its local policy, a domain may simultaneously (or asynchronously) change these $PIDs$. In the former case, the cost for updating the $PIDs$ is fixed since a domain only needs to distribute the new $PIDs$ to its border routers once every PID update period. In the latter case, every time the PID of an inter-domain path is updated, the domain needs to distribute the new PID to its border routers. As it is shown in Section V-C, however, the cost for updating $PIDs$ in the latter case is significantly less than the update cost of IP-prefixes in the Internet today.

B. Negotiating PIDs

Since inter-domain packet forwarding is based on $PIDs$, it is necessary to guarantee that the $PIDs$ used by a domain are different from each other, even if they change dynamically. To achieve this, a direct approach is for domain A to notify its neighboring domain B the set of $PIDs$ that are used (or conversely, the set of $PIDs$ not used) by domain A, and domain B chooses a PID not used by both domains A and B. However, domains may be reluctant to adopt this approach since it may leak their privacy. More importantly, a domain can have as much as 5,000 neighboring domains [33] and may simultaneously negotiate $PIDs$ for paths that connect the domain with these neighbors. As a result, two neighboring domains of a domain A may simultaneously choose a common

PID for two paths. This in turn entails multiple rounds of negotiation, leading to a very long negotiation delay.

Therefore, we propose an approach that 1) guarantees the negotiated $PIDs$ are unique; and 2) requires only one round of negotiation for each path. To achieve this, two domains negotiate a PID block represented by a PID-prefix (like an IP prefix) to every inter-domain path between them at the bootstrapping stage when they interconnect with each other. The $PIDs$ belonging to the PID-prefix are then only used to represent the path. The PID-prefix assigned to an inter-domain path is unique in the sense that the two domains connected by the path do not assign the PID-prefix to any other inter-domain path. This means that a PID-prefix could be used to represent multiple inter-domain paths in the Internet. For instance, the PID-prefix 0x1B000000/8 is assigned by domains A and C to inter-domain path P_2 in Fig. 4. At the same time, it is also assigned by domains B and E to inter-domain path P_4 in Fig. 4. This way, it is scalable for domains to assign PID-prefixes since it is not necessary for domains to globally cooperate to assign PID-prefixes to inter- domain paths.

As IPv4 addresses, $PIDs$ are 32-bit long in our implementation. First, the largest AS in the current Internet has about 5,000 neighboring domains [33]. Since two domains may have multiple (typically, up to four) inter-domain paths [34] and every path needs a unique PID-prefix, the largest AS may need about 20,000 PID-prefixes, which entails 15 bits to represent. Second, the number of $PIDs$ within a PID-prefix should not be too small in order to prevent attackers from correctly guessing the PID used by a path. Assume that an ingress border router can detect an attack and alarm the neighboring domain to suppress the attack if it receives ten data packets carrying wrong $PIDs$ every second from a neighboring domain. Accordingly, an attacker cannot send more than $20 \times T$ data packets carrying wrong $PIDs$ during a PID update period T in order to avoid being detected, since a PID needs to be used to forward data packets for two PID update period T. Even if T is set to be 3,600 seconds, an attacker can try 72,000 $PIDs$ in a PID update period. Accordingly, it is enough to use 17 bits to present the number of $PIDs$ within a PID-prefix. Third, $PIDs$ should not be too long since using longer $PIDs$ consumes more network bandwidth.

For ease of presentation, we use hexadecimal numbers to denote $PIDs$ and PID prefixes. Similar to current IP addresses, we also use a PID-prefix mask to denote the length of a PID-prefix. For example, a PID can be represented as “0x1A01010A/8”, where “/8” represents the mask length of the PID (*i.e.*, eight bits). In Fig. 4, the path P_1 connecting domains A and B is assigned the PID-prefix 0x1A000000/8, and the path P_2 connecting domains A and C is assigned the PID-prefix 0x1B000000/8. Similarly, other inter-domain paths are also assigned corresponding PID-prefixes, as shown in Fig. 4. Note that the PID blocks assigned to paths may be of different size. For instance, two domains may assign a /8 PID-prefix to an inter-domain path, and a /4 prefix to another one connecting them.

The actual negotiation process is quite simple. Recall that, for the PID of a path connecting two neighboring domains, it is associated with an update period. At the end of the update period, the initiative RM in the two neighboring domains randomly chooses a new PID from the PID-prefix assigned to the path, and sends the chosen PID to the RM in another domain, as illustrated by (1) in Fig. 4. If the later one accepts the chosen PID, it sends a confirmation message back to the initiative RM, as illustrated by (2) in Fig. 4. Otherwise, the later RM chooses another PID from the PID-prefix and sends the chosen PID back to the initiative RM. As stated in Sec. III-A, if the update period of a path is not fixed, the two domains can also negotiate the new period used to update the PID of the path.

C. Distributing PIDs to Routers

Having negotiated the new PID of a path connecting a neighboring domain, the RM in a domain A needs to distribute the new PID to the routers in that domain so that the new PID can be used to forward data packets. To achieve this, the RM simply sends a PID update message to every border router in the domain. The PID update message contains the path and its corresponding new PID. When a border router receives the PID update message, it updates its inter-domain routing table. After that, it sends an acknowledgement message to the RM. When the RM receives the acknowledgement messages from all border routers, it then updates its PID table. After that, the RM appends the new PID instead of the old one onto the GET messages when it forwards them. Accordingly, the corresponding data packets will be forwarded from the neighboring domain to domain A by using the new PID.

Fig. 4 illustrates the PID distribution process, assuming that domains A and B change path P_1's PID from 0x1A01010A to 0x1AF12C03 during the PID negotiation process. In addition, before path P_1's new PID is distributed, the inter-domain routing table of router R_2 is shown at the bottom left corner in Fig. 4. Note that in order to maintain legal communications (Sec. III-D), the inter-domain routing table of router R_2 has two entries for path P_1: 0x1A01010A that is used by the RM before the negotiation and will be replaced by 0x1AF12C03 after the negotiation, and 0x1AE581FA that is used in the previous negotiation and has been replaced by 0x1A01010A. Similarly, after the negotiation is completed, the PID table of the RM is shown at the upper left corner in Fig. 4. When the negotiation completes, RM_1 distributes the new PID 0x1AF12C03 of path P_1 to the border routers in domain A by sending to each of them a PID update message, as illustrated by (3) in Fig. 4. When border router R_2 receives the PID update message, it deletes the outdated PID 0x1AE581FA and inserts the new PID 0x1AF12C03 into its inter-domain routing table, as illustrated by (4) in Fig. 4. Having updated its inter-domain routing table, router R_2 then sends an acknowledgement message to its local RM (*i.e.*, RM_1), as illustrated by (5) in Fig. 4. When RM_1 receives the acknowledgement messages from all border routers, it updates its PID table, as illustrated by (6) in Fig. 4. At this time, the PID distribution process completes and RM_1 forwards GET messages to RM_2 by using the new PID 0x1AF12C03 instead of the old 0x1A01010A, as illustrated by (7) in Fig.

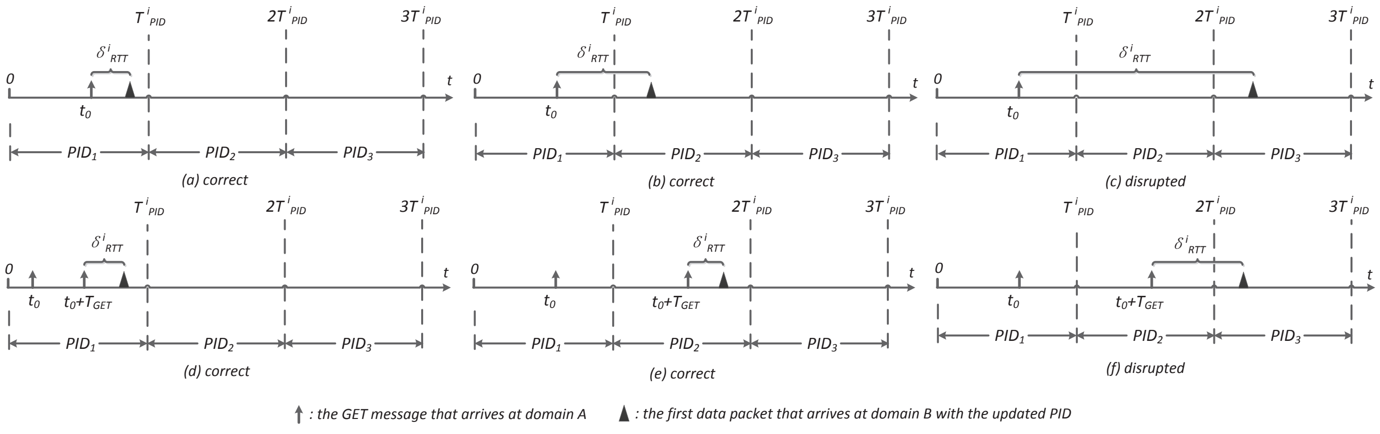

(a) correct
(b) correct
(c) disrupted
(d) correct
(e) correct
(f) disrupted
T^i_{PID} $2T^i_{PID}$ $3T^i_{PID}$
δ^i_{RTT}
t_0 t_0+T_{GET}
PID_1 PID_2 PID_3
: the GET message that arrives at domain A
: the first data packet that arrives at domain B with the updated PID

Fig. 5. The mathematical model for determining T_{PID} and T_{GET}.

4. Similarly, RM_2 also distributes the new PID to the border routers in domain B.

It is worthy of noting that the RM will update its PID table after it receives acknowledgements from all border routers so that the border routers do not discard data packets that carry the new PID (e.g., 0x1AF12C03). Since the RM and border routers are in the same domain, the delay used for waiting the acknowledgements from border routers is fairly small (less than one second), especially when compared with the PID update period (several tens of seconds).

D. Maintaining Legal Communications

As mentioned before, the communications that last long enough (*e.g.*, longer than $2T_{PID}$) may be interrupted when D-PID is applied in CoLoR. As a result, we need to design a mechanism to timely update the $PIDs$ that are used by the active communication between a content provider and a content consumer. To achieve this, we let the subscriber periodically retransmit the same GET message for an active communication. When the source receives the PID sequence contained by the retransmitted GET message, it updates the old PID sequence to the fresh one and uses the updated PID sequence to send subsequent data packets. Accordingly, the data packets from the source can be correctly forwarded to the subscriber and the ongoing communication will not be interrupted. In the rest of this paper, we denote the GET retransmission period as T_{GET}.

We use the example in Fig. 4 to further illustrate such a process. Assuming that, before time t_1, the subscriber in domain C sends a GET message to the source in domain F to request a content. Thus, the source initially sends the corresponding data packets back to the subscriber based on the old PID 0x1A01010A. After t_1, the PID is updated to 0x1AF12C03, while the data plane can still forward data packets based on the old PID 0x1A01010A. Accordingly, the communication will not be interrupted before $t_1 + T_1$, but will be interrupted after $t_1 + T_1$. However, if the subscriber timely retransmits the same GET message to the source (an enough time period before $t_1 + T_1$), the new PID 0x1AF12C03 will be encapsulated in the GET message and sent to the source. This way, the source knows the new PID 0x1AF12C03 and uses it to forward the subsequent data packets to the subscriber. Therefore, if the subscriber periodically retransmits the GET message and the period is appropriately set, the source can know the valid $PIDs$ toward the subscriber, thus maintaining a legal communication.

It is worthy of noting that the retransmission of a GET message is initiated by a content consumer based on its GET retransmission period T_{GET} and the initial time the consumer sends out the first GET message for a piece of content. Therefore, when the PID of an inter-domain path from the content provider to the consumer changes, the content consumer does not resend a GET message immediately when the $itPID$ changes. This way, we can efficiently avoid the sudden increase in the number of GET messages received by RMs in the Internet, as it is shown in Fig. 12.

E. Setting $T_{\mathbf{PID}}$ and $T_{\mathbf{GET}}$

In D-PID, T_{PID} and T_{GET} should be carefully set so that a legal communication will not be interrupted when $PIDs$ change dynamically. To achieve this, we build a mathematical model to calculate the appropriate values of T_{PID} and T_{GET}, with the help of Fig. 5.

Without loss of generality, we assume that there are N inter-domain paths along the path from a server to a client. In particular, we consider the $i-th$ $(1 \leq i \leq N)$ inter-domain path and assume that it connects two domains A and B. Specifically, we call the timeout period in which a GET message arrives at domain A as the present timeout period and the GET message will be forwarded to domain B. We also assume that the present timeout period begins at time zero, as illustrated by Fig. 5. In addition, we assume that the GET message arrives at domain A at time t^i_0 and the round-trip time from domain A to the content provider is δ^i_{RTT}. For ease of presentation, we denote the timeout period of the $i-th$ path be T^i_{PID}.

With these assumptions, one can observe from Fig. 5(a) that, if $(t^i_0 + \delta^i_{RTT})$ is less than T^i_{PID}, the corresponding data packets for the GET message will arrive at domain B in the same timeout period in which the GET message arrives

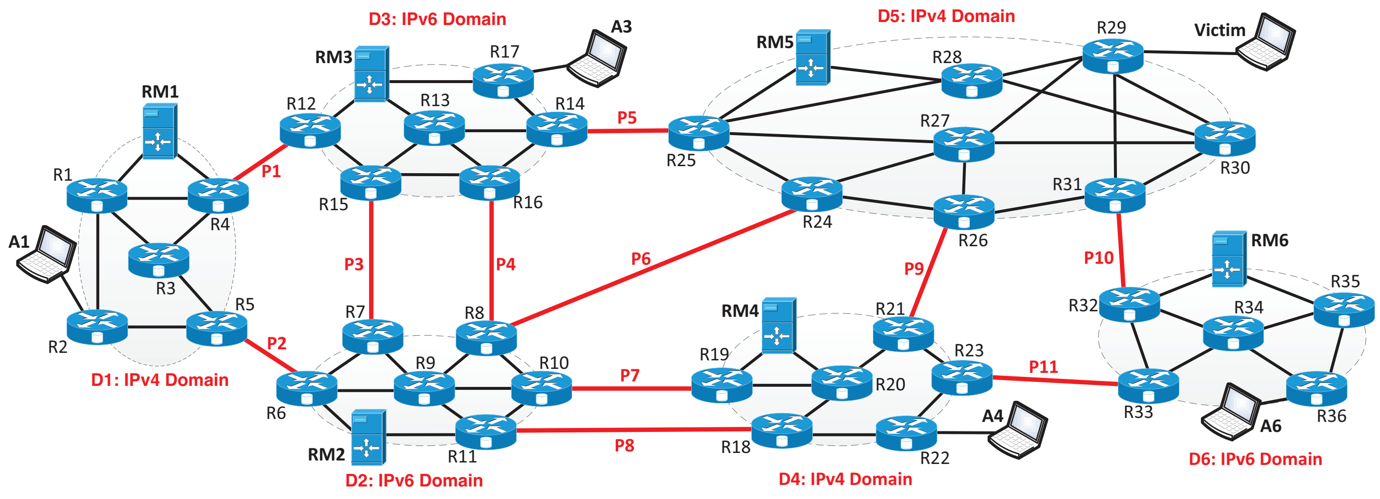


Fig. 6. The topology of the prototype.

at domain A. In this case, the data packets can be correctly forwarded to domain A. Similarly, if $(t_0^i + \delta_{RTT}^i)$ is less than $2T_{PID}^i$, the corresponding data packets will arrive at domain B in the next timeout period, as shown by Fig. 5(b). In this case, the data packets also can be correctly forwarded to domain A because domain B is able to forward data packets based on the $PIDs$ chosen for the present and the previous timeout periods.

However, when $(t_0^i + \delta_{RTT}^i)$ is larger than $2T_{PID}^i$, as shown in Fig. 5(c), data packets will be dropped by domain B. Therefore, in order to guarantee correct data packet forwarding, it must hold that:

$$t_0^i + \delta_{RTT}^i < 2T_{PID}^i.$$

Note that t_0^i should be evenly distributed in $[0, T_{PID}^i)$. To guarantee correct packet forwarding, it requires that:

$$T_{PID}^i > sup(\delta_{RTT}^i),$$

where $sup(x)$ represents the supremum of x. This formula indicates that when we set T_{PID} for an inter-domain path, the value should be greater than the supremum value of the network's round-trip time. From Fig. 13 in [35], we know that with probability higher than 99.9%, the round-trip time is less than 1.5 seconds. Therefore, we can treat the value of $sup(\delta_{RTT}^i)$ as 2 seconds in practice.

We now describe how to set the value of the GET retransmission period T_{GET} for an active session. Obviously, the second GET message (i.e., the first retransmitted GET message) arrives at domain A at the time $(t_0^i + T_{GET})$. When the data source receives the second GET message, it will renew the PID sequence and then sends subsequent data packets to the client by using the new PID sequence. We assume that the first one of these subsequent packets arrives at domain B at time $(t_0^i + T_{GET} + \delta_{RTT}^i)$.

Similar to our discussions on setting T_{PID}^i, there are also three cases. In the first case, t_0^i and $(t_0^i + T_{GET} + \delta_{RTT}^i)$ are in the same timeout period, as illustrated by Fig. 5(d). In this case, when domain B receives the data packets sent by the server, it can correctly forward these data packets to domain A. In the second case, $(t_0^i + T_{GET} + \delta_{RTT}^i)$ is in the next timeout period to t_0^i, as illustrated by Fig. 5(e). In this case, domain B also can correctly forward data packets to domain A because now domain B can forward packets based on both PID_1 and PID_2. In the third case, $(t_0^i + T_{GET} + \delta_{RTT}^i)$ is larger than $2T_{PID}^i$, as illustrated by Fig. 5(f). In this case, some data packets will be discarded during the period $(2T_{PID}^i$, $t_0^i + T_{GET} + \delta_{RTT}^i)$. Therefore, to guarantee the correct data forwarding, it must hold that:

$$t_0^i + T_{GET} + \delta_{RTT}^i < 2T_{PID}^i.$$

As discussed before, t_0^i may be very close to T_{PID}^i, so the above inequation can be rewritten as:

$$T_{GET} < T_{PID}^i - sup(\delta_{RTT}^i).$$

Note that the above discussions are focused on a given inter-domain path. Since there are N inter-domain paths between the client and the server, to ensure that the data packets could be correctly forwarded along all intermediate paths, the T_{GET} should be given by:

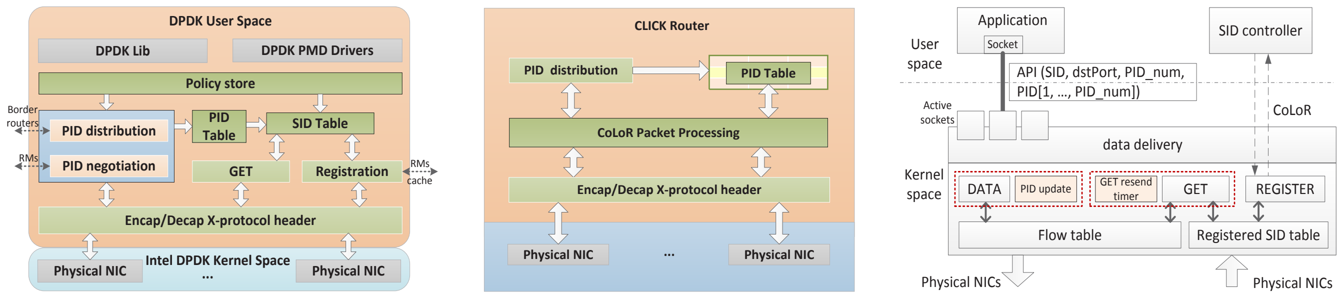
$$T_{GET} < \min_{i=1,2,\ldots,N} (T_{PID}^i - sup(\delta_{RTT}^i)).$$

Without loss of generality, we assume that $sup(\delta_{RTT}^i)$ is the same for all inter-domain paths. In addition, as discussed before, we can set $sup(\delta_{RTT}^i)$ to be 2 seconds in practice. Accordingly, the above inequation can be rewritten as:

$$T_{GET} < \min_{i=1,2,\ldots,N} (T_{PID}^i) - 2. \tag{1}$$

F. Collecting Minimum $T_{\mathbf{PID}}^{\mathbf{i}}$

From Equation (1), we know that the subscriber needs to be aware of $min_{i=1,2,\cdots,N}(T_{PID}^i)$ in order to appropriately set T_{GET}. To achieve this, we add a field called *MINIMUM_PERIOD* in the GET message and the data packet to collect the PID update period T_{PID}^i of $PIDs$ along the path from the subscriber to the source. When the subscriber sends out a GET message, the value of *MINIMUM_PERIOD* is set to be infinite. When a RM receives a GET message and chooses



(a) *RMs* (b) Border routers (c) End hosts

Fig. 7. The structures of RMs, border routers, and end hosts in our implementation.

the inter-domain path toward the next hop domain, it compares the value of *MINIMUM_PERIOD* with the path's T_{PID}. If the path's T_{PID} is smaller than the value of *MINIMUM_PERIOD*, the RM replaces the value of *MINIMUM_PERIOD* by the path's T_{PID}. When the source receives the GET message, it writes the value of *MINIMUM_PERIOD* into the data packets that are sent to the content subscriber. This way, the content subscriber knows the minimum of all T^i_{PID} along the path from the content subscriber to the content provider.

IV. Prototype Implementation

We verified D-PID's feasibility and effectiveness by implementing it in a 42-node prototype. Our implementation effort was instrumental in refining our design, leading to several revisions. For example, we initially use the approach discussed in the first paragraph in Sec. III-B to negotiate $PIDs$. Through experiments, we found such a design may cause multiple rounds of PID negotiations, which stimulated us to propose the PID-prefix approach described in Sec. III-B. Below we describe our implementations and present results from running experiments on the prototype.

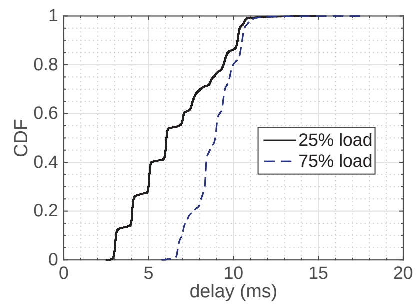
A. Prototype Design

The prototype has six domains (*i.e.*, D_1 - D_6) that use different intra-domain routing protocols, as shown in Fig. 6. The six domains are inter-connected by 11 inter-domain paths (*i.e.*, P_1 - P_{11}) , each of which is assigned with a PID-prefix based on the design in Sec. III. Every domain has one centralized RM. Every node in the prototype (including the routers, the RMs, and the end-hosts) is running on an aTCA-9300 processor blade, with a four-core Intel Xeon E3 1275V2 processor, an 8 GB DDR3-1600 memory, and six Intel I210 Gigabit Ethernet controllers. The RMs are implemented based on the DPDK [36] platform for fast packet processing, the routers are implemented by using the CLICK software platform [37], and the end-hosts are implemented as a module in Linux kernel version 2.6.35. We now present the implementation details of the prototype.

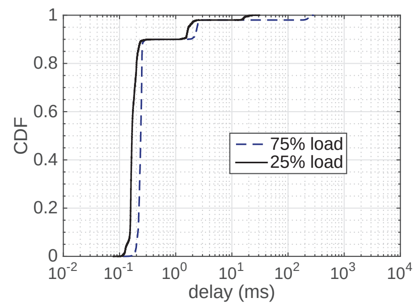
1) RMs: Fig. 7 (a) shows the structure of the implemented RMs, where "X-protocol" represents the local routing protocol used by the domain where the RM locates. The Registration module is used to process registration messages, and it stores the reachability information of the registered content names into the SID Table. The GET module is used to process GET messages, and it queries the SID Table in order to determine the next hop for a GET message. The PID Table stores the currently used $PIDs$ for the inter-domain paths associated with the domain where the RM locates. To support D-PID, an entry in the PID table has a timer recording the time that a new PID should be negotiated. When the timer of a PID entry times out, the PID negotiation module negotiates a new PID for the inter-domain path with the associated neighbor RM. When the negotiation completes, the PID distribution module distributes new $PIDs$ to border routers in a domain.

2) Border Routers: Fig. 7 (b) shows the structure of the implemented border routers, where "X-protocol" represents the local routing protocol used by the domain where the border router locates. The Packet Processing module is used to process CoLoR format packets based on the $PIDs$, and it queries the PID Table to determine the operation for an incoming packet (*e.g.*, encapsulating the packet with an IPv4 packet header and sending it to another border router). The PID distribution module is used to process PID update messages from the RM. When it receives a PID update message, it adds the new PID into the PID table and sends an acknowledgement back to the RM. In addition, a PID entry in the PID table also has a timer recording the time that the PID should be removed from the PID table. Once the timer of a PID entry in the PID table expires, the entry is deleted from the PID table.

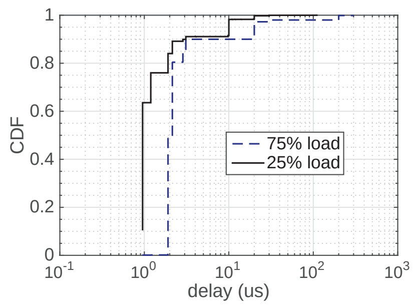
3) End Hosts: Fig. 7(c) shows the structure of the implemented end hosts. We implement CoLoR as an independent protocol stack (as same as the TCP/IP stack) in the Linux kernel, and provide APIs (Application Program Interfaces) for applications to call the CoLoR socket that can send/receive GET, data, and registration messages. In particular, we embed several functionalities into the CoLoR stack in the Linux kernel. To collect the minimum T_{PID}, the DATA module reads the *MINIMUM_PERIOD* field when it receives a data packet, and sets the timer to resend GET messages for the associated session based on *MINIMUM_PERIOD*. When the timer for the session times out, the GET module re-sends the GET message to the content provider in order to refresh the $PIDs$. When the source receives a resent GET message for an active session, the PID update module refreshes the PID sequence used by the session based on the $PIDs$ contained in the GET message.



(a) The PID negotiation delay



(b) The PID distribution delay



(c) The PID installation delay

Fig. 8. The PID negotiation performance obtained from the prototype.

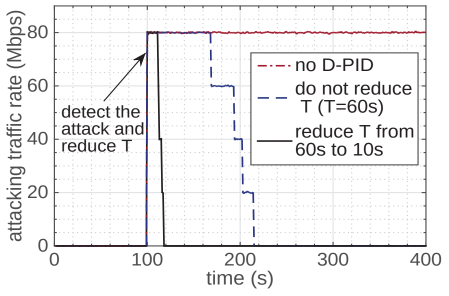
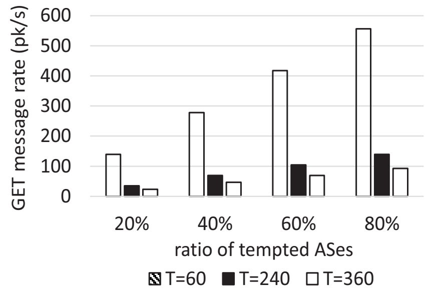
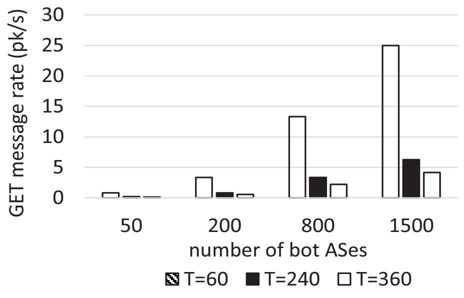


Fig. 9. The attacking rate received by the victim.



(a) The GET luring



(b) The botnet cooperation

Fig. 10. The GET message rates received by attackers.

B. Testbed Experiments

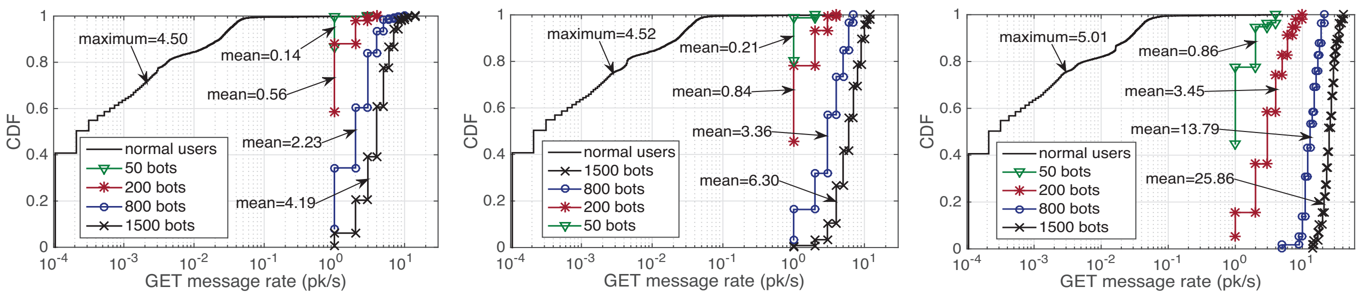
To evaluate the performance of D-PID, we tested the PID negotiation delay, the PID distribution delay, and the PID installation delay in our prototype. The PID negotiation delay is the interval between the time when a RM sends the PID update message to its neighbor RM and the time when it receives the update confirmation message from its neighboring RM (*i.e.*, the duration of steps (1) and (2) in Fig. 4). The PID distribution delay is the interval between the time when a RM sends the PID distribution message to a border router and the time when the RM receives the confirmation message from all border routers (*i.e.*, the duration of steps (3), (4), and (5) in Fig. 4). The PID installation delay is the time for a border router to install a PID into its forwarding table (*i.e.*, step (4) in Fig. 4). We also record the three kinds of delays in two different scenarios: 1) the border routers are forwarding data packets at 25% link speed, and 2) the border routers are forwarding data packets at 75% link speed. For each kind of delay, we record 20,000 samples by running D-PID in the prototype.

Fig. 8(a) shows the cumulative distribution function (CDF) of the PID negotiation delay. We observe from Fig. 8(a) that all PID negotiations are finished within 20 milliseconds (ms). Fig. 8(b) and Fig. 8(c) show the $CDFs$ of the PID distribution delay and the PID installation delay, respectively. From them, we observe that the PID distribution delay is less than 3 ms with a probability higher than 99% and the PID installation is less than 20 microseconds (us) with a probability higher than 99%. In addition, the traffic load of the border routers does not have much effect on the PID distribution delay and the PID installation delay. From Fig. 8, we conclude that the PID distribution delay is mainly determined by the maximal end-to-end delay from the RM to the border routers.

C. Resilient to DDoS Flooding Attacks

We conduct an experiment to test the effect of D-PID on defending against DDoS flooding attacks. To achieve this, we set four attackers (*i.e.*, A_1, A_3, A_4, and A_6 in Fig. 6) and a victim in domain D_5, assuming that the attackers have learned the $PIDs$ in the network before the attack. At time 100 second, we let each attacker flood the victim at a rate of 20 Mbps (megabits per second) and observe the attacking traffic at the victim. There are three different scenarios in our experiments: 1) the $PIDs$ are static; 2) the $PIDs$ dynamically change with an update period of 60 seconds; and 3) the update period of the $PIDs$ is 60 seconds at the beginning, and when domain D_5 detects the attack, it changes the update period of the $PIDs$ P_{10}, P_9, P_6, and P_5 to 10 seconds.

Fig. 9 shows the observed attacking traffic at the victim in the three scenarios. From the results, we observe that the attacking traffic is 60 Mbps if D-PID is not applied. On the other hand, the attacking traffic is gradually eliminated (in 115 seconds) if the $PIDs$ dynamically change once every 60 seconds. Furthermore, if the network (*i.e.*, D_5) where the victim locates reduces the PID update period when it detects the attack at 100 second, the attacking traffic is quickly removed (in only 18 seconds). This indicates that D-PID not only can effectively defend against DDoS attacks, but also can make it possible for the victim to actively take measures to eliminate an ongoing attack. Note that if the attacker periodically learns the $PIDs$ in the network, the attacking traffic cannot be eliminated. However, this not only significantly increases the cost for the attacker to launch attacks, but also makes it easier



(a) T_{PID} = *360* seconds (b) T_{PID} = *240* seconds (c) T_{PID} = *60* seconds

Fig. 11. The GET message rate of normal users and attackers in the botnet cooperation.

for the network to detect the attacker, as it is shown in Sec. V-A.

V. Simulation Results

In this section, we first evaluate the effect of D-PID in defending against DDoS attacks in large scale networks. We then evaluate D-PID's overheads, including the extra GET messages sent by the subscriber, and the control overhead incurred by PID negotiation and distribution.

A. Defending Against DDoS Attacks

1) Attacking Cost: In D-PID, the $PIDs$ learned by the attacker at the PID learning stage (as described in Sec. II) will become invalid after a certain period. As a result, if the attacker does not continuously (or periodically) learn the valid $PIDs$ in the network, the attacking traffic rate received by the victim will be quickly reduced to zero, as demonstrated in Sec. IV-C. On the contrary, if the attacker periodically learns the valid $PIDs$ in the network, it will receive more GET messages, thus increasing the attacking cost. Below we evaluate such a cost through extensive simulations.

We use the AS-level topology of the current Internet, which is derived from the BGP data set collected by Route Views, RIPE RIS, PCH, and Internet2 [38]. The data set has 49,448 $ASes$ and 212,543 AS links. We divide all $ASes$ in the topology into two categories: edge $ASes$ and core $ASes$. An AS is classified as an edge AS if it always appears at the end point of an AS path [39]. As a result, we identify 41,734 edge $ASes$ and 7,714 core $ASes$. In all simulations, we assume that the attacking $ASes$ are chosen from the edge $ASes$.

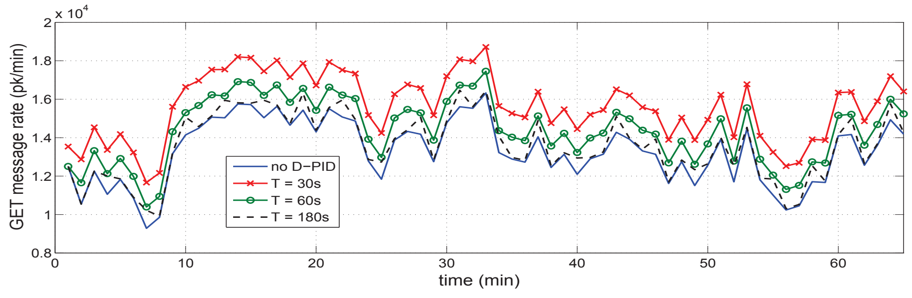
For GET luring, we randomly choose an edge AS as the luring AS and let it publish content names into the network. Then we choose another portion (*i.e.*, 20%, 40%, 60%, and 80%) of edge $ASes$ as tempted $ASes$ and let them send GET messages to the luring AS at the PID learning stage so that the attacker can obtain some $PIDs$ in the network. In the attacking stage, we randomly choose an edge AS that is along the learned path as the victim AS, and a portion of edge $ASes$ (along the learned path) as attacking $ASes$. To launch the attack, we let the luring AS periodically learn the $PIDs$ in the network, and record the GET message rate (*i.e.*, the number of GET messages received by the attacker per second).

For the botnet cooperation, we randomly choose a number of edge $ASes$ (50, 200, 800, and 1,500, respectively) as bot $ASes$, and let them send GET messages to each other. The maximum number of bot $ASes$ is set to be 1,500 in the simulations because the number of source $ASes$ in almost all current DDoS attacks is under 1,000 [40]. Then we randomly choose an edge AS along the learned path as the victim AS. To launch attacks, we let the bot $ASes$ periodically learn the $PIDs$ in the network and record the average GET message rate received by all bot $ASes$. For each combination of the parameters, we run the simulation 20 times. All results shown below are the mean values of 20 simulations.

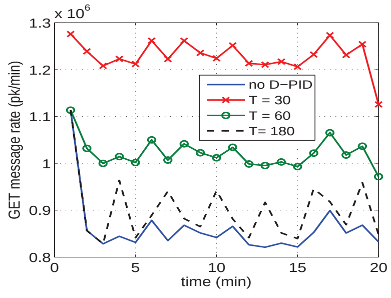
Fig. 10(a) shows the GET message rate that a luring AS receives in GET luring, if D-PID is deployed and the attacker still tries to learn the valid $PIDs$ in the network. From Fig. 10(a), we observe that the PID update period (T in Fig. 10) has a significant effect on the attacking cost. For example, when the ratio of tempted $ASes$ is 80% and the PID update period is 360 seconds, a luring AS receives less than 100 GET messages per second. However, when the PID update period is 60 seconds, a luring AS will receive about 570 GET messages per second. From Fig. 10(a), we also observe that, when T = 60, an attacker receives about 150 GET messages per second if he controls only 20% $ASes$. However, if the attacker controls 80% $ASes$, he receives as high as 570 GET messages per second. Therefore, the more $ASes$ controlled by an attacker, the more GET packets the attacker receives. This not only hurts the attacker itself since it receives more traffic, but also makes it easier for the attacker to be detected by using the number of GET messages it receives. By contrast, if $PIDs$ do not change, the attacker will not receive GET messages once he has learned the $PIDs$ in the network.

Fig. 10(b) shows the GET message rate that a bot AS will receive in the botnet cooperation, if D-PID is deployed and the attacker still tries to continuously learn the valid $PIDs$ in the network. From Fig. 10(b), we can make conclusions similar to those from Fig. 10(a). Therefore, D-PID can significantly increase the cost in launching DDoS attacks, which cannot be achieved if the $PIDs$ are static.

2) Detecting DDoS Attacks: With D-PID, an attacker has to periodically learn the valid $PIDs$ to launch a DDoS flooding attack, thus receiving a large amount of GET messages. This in turn provides a new dimension for the network to detect the attackers, since a normal user in the Internet usually receives very few GET messages. To illustrate this, we conduct emulations based on the real data trace collected from a



(a) Datacenter trace



(b) Tier-1 AS traces

Fig. 12. The GET message rates obtained from the data traces.

university campus [41]. In the emulations, we assume that when a node opens a new TCP/IP flow to another node, it receives a GET message. In addition, before the flow is finished, the node also receives a GET message at the end of every GET retransmission period.

In the emulations, we record the GET message rates received by 6,601 normal users in the data trace for 9,481 seconds, and calculate the mean GET message rate received by each user during the 9,481 seconds when the PID update period T_{PID} is 30s, 180s, and 300s, respectively. For comparison, we also simulate (as described in Sec. V-A1) the GET message rate received by the attackers in the botnet cooperation when the PID update period T_{PID} is 60s, 240s, and 360s, respectively, assuming that the number of bot $ASes$ controlled by the attacker is 50, 200, 800, and 1,500.

Fig. 11(a) shows the empirical CDF of the GET message rates of attackers and normal users when the PID update period T_{PID} is 360s. From Fig. 11(a), we observe that the number of GET messages received by the attacker is significantly larger than that received by most of the normal users, regardless how many bot $ASes$ the attacker controls. This implies that we can count the GET message rates of users to detect DDoS attacks. While there are still some normal users (*e.g.*, web servers) receiving more GET messages per second than attackers, we can narrow our focus to the users who receive significantly more GET messages per second and try to identify attackers among those users. To this end, we can simply reduce the PID update period T_{PID} and observe the changes in the GET message rates of them. For instance, when the PID update period T_{PID} reduces from 360 seconds to 240 seconds, the GET message rate of the attacker increases from 0.56 to 0.84 (*i.e.*, significantly increased by 50%) if the attacker controls only 200 bot $ASes$. By contrast, the GET message rate of a normal user (with the max GET message rate) only increases from 4.50 to 4.52 (*i.e.*, slightly increased by 0.4%). When the PID update period further reduces to 60 seconds, the GET message rate of the attacker increases from 0.84 to 3.45 (*i.e.*, increased by 310.7%), but that of a normal user only increases from 4.52 to 5.01 (*i.e.*, increased by 10.8%). This way, we can effectively detect the attacker.

B. Extra GET Messages

Applying D-PID in CoLoR requires that the content consumer should periodically retransmit GET messages. This in turn causes extra overhead on the network, especially on the RMs. We evaluate such overheads by using two sets of real data traces. The first real data trace is collected from a data center and lasts 65 minutes [41]. The second trace lasts 20 minutes and is collected from a Tier-1 link [42]. For our evaluation, we develop a packet-level simulator based on C++, which could simulate the generation of GET massages based on the active flows in the data traces, under different values of T_{GET}. In particular, we define a flow as a sequence of packets that share the same 5-tuple $\langle$*the source IP address, the destination IP address, the source port number, the destination port number, the protocol*$\rangle$. We count the number GET messages sent by the content consumers under different T_{GET}.

TABLE I
THE MEAN GET MESSAGE RATE (NO. OF GET MSG. PER SECOND)

T_{GET}	30s	60s	180s	300s	600s	no D-PID
DC	260.4	239.7	226.9	224.6	222.9	221.5
Tier-1	20481	17027	14994	14662	14432	14348

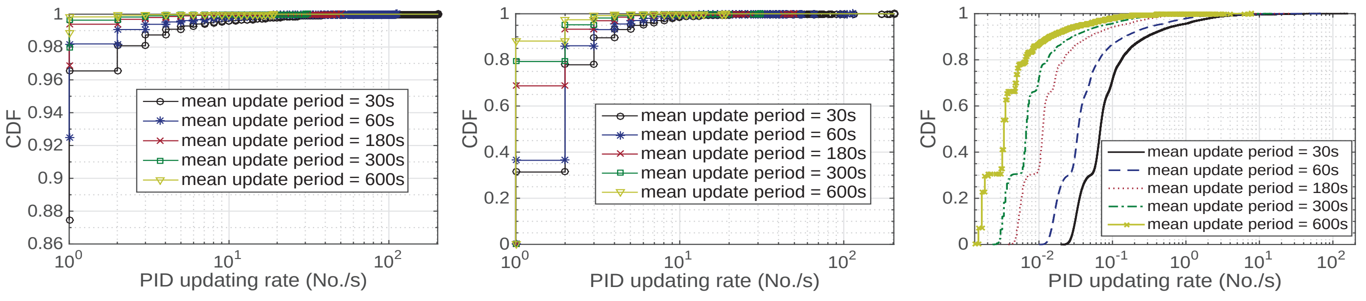
For the data center data trace, Fig. 12(a) shows the number of GET messages sent by the content consumers per minute when T_{GET} is 30, 60, and 180 seconds, respectively. For comparison, we also show the number of GET messages sent by the content consumers per minute when we do not deploy D-PID (represented by "no D-PID"). Table I shows the average number of GET messages per second for different T_{GET}. From the results, we observe that the extra number of GET messages is about 8.2% ($= \frac{(239.7-221.5)}{221.5}$) when T_{GET} is 60 seconds. When T_{GET} increases to 300 seconds, however, this number is reduced to about only 1.4% ($= \frac{(224.6-221.5)}{221.5}$) .

Fig. 12(b) and Table I show the number of GET messages sent by the content consumers per minute when T_{GET} is 30, 60, and 180 seconds, respectively, for the Tier-1 network data trace. From them, we observe that the extra number of GET messages is about 18.7% ($= \frac{(17027-14348)}{14348}$) when T_{GET} is 60 seconds. However, this number is reduced to about only 2.2% ($= \frac{(14662-14348)}{14348}$) when T_{GET} increases to 300 seconds.

In summary, these results indicate that the extra number of GET messages caused by the D-PID mechanism could be trivial if T_{GET} is properly set.

C. Control Overhead

To evaluate the control overhead caused by PID negotiation and distribution, we conduct simulations with the real Internet



(a) Total PID update rates (b) Peak PID update rates (c) Mean PID update rates

Fig. 13. Empirical cumulative density function (CDF) of PID update rates.

topology [38] used in Sec. V-A. In the simulations, we assign each inter-domain path with a PID update period, and all the periods are normally distributed. We conduct the simulations by using different parameters: the mean value and the standard variance of these periods is set to be (30s, 5s), (60s, 10s), (180s, 20s), (300s, 30s), and (600s, 40s), respectively. At the beginning of the simulations, for each inter-domain path, we let the value of its first PID update period be uniformly distributed between zero and its PID update period. During the simulations, once the PID of an inter-domain path needs to be updated, we record the time and the two domains associated with the path. Therefore, we have the PID update rate for every domain in the topology. All simulations are conducted for 6,000 seconds.

Fig. 13 shows the CDF of the PID update rates of all domains per second, the CDF of the peak PID update rates of every domain, and the CDF of the mean PID update rates of every domain, respectively, when the mean value of the update period is set to different values. From the results, we can observe that even if the mean PID update period is 30s, the PID update rate is less than 10 per second with a probability higher than 99% (Fig. 13(a)). Moreover, the peak PID update rate per domain is less than 10 per second with a probability higher than 95% (Fig. 13(b)), and the mean PID update rate per domain is less than one per second with a probability higher than 95% (Fig. 13(c)). Note that when the mean PID update period is 30s, the maximal PID update rate in the network is 202 per second. We can also observe that, when we increase the mean PID update period, the PID update rate in the network is significantly decreased. For example, if the mean PID update period is 600s, the peak PID update rate per domain is less than one per second with a probability higher than 87%, and the maximal PID update rate in the network is only 19 per second (Fig. 13(b)).

Note that the PID update rate is significantly less than the update rate of IP-prefixes in the current Internet. First, the peak update rate of IP-prefixes is 1,962 per second (at 15:26:01, March 14, 2015) [32], while the peak update rate of $PIDs$ is only 202 per second when the mean PID update period is 30s. Second, the mean update rate of IP-prefixes is 5.23 per second [32], while the mean update rate of $PIDs$ is only 0.295 per second and 0.014 per second when the PID update periods are 30s and 600s, respectively. Third, all Tier-1 domains suffer from high update rate of IP-prefixes (up to 1,962 per second) in the current Internet, while only a few large domains having thousands of neighbors suffer from high PID update rate (near to 202 per second) in D-PID and the vast majority of domains process trivial number of PID updates (Fig. 13).

VI. Related Work

Because of the complexity and difficulty in defending against DDoS flooding attacks, many approaches have been proposed in past two decades. For instance, filtering-based approaches aim at mitigating DDoS flooding attacks by deploying source address filtering at routers [5] - [9]. Similarly, IP traceback-based methods trace attacks back through the network toward the attacking sources [10] - [14]. In addition, the approaches proposed in [19] - [20] aim at mitigating DDoS attacks by sending shut-up messages to the attacking sources, assuming that they will cooperate and stop flooding. While there are too many literatures, we refer interested readers to [4] for a survey on existing approaches in defending again DDoS flooding attacks. Instead, we outline prior work closely related to this work and compare D-PID with them.

A main reason that DDoS flooding attacks proliferate is a node can send any amount of data packets to any destination, regardless whether or not the destination wants the packets. To address this issue, several approaches have been proposed. In the "*off by default*" approach [15], two hosts are not permitted to communicate by default. Instead, an end host explicitly signals, and routers exchange, the IP-prefixes that the end host wants to receive data packets from them by using an IP-level control protocol. The D-PID design is similar in spirt, since D-PID dynamically changes $PIDs$ and a content provider can send data packets to a destination only when the destination explicitly sends out a GET message that is routed (by name) to the content provider. However, there are two important differences. First, the "off by default" approach works at the IP-prefix granularity, but D-PID is based on an information-centric network architecture and works at the content granularity. Second, the IP-prefixes that an end host wants to receive packets from are propagated throughout the Internet in the "off by default" approach, which may cause significant routing dynamics if the allowed IP-prefixes of end hosts change frequently. On the other hand, the $PIDs$ are kept secret and change dynamically in D-PID. While this incurs cost since destinations need to re-send GET messages, the results presented in Sec. V show that the cost is fairly small.

The capability-based designs [16] - [17] also share the same spirt with "off by default" and D-PID. In these approaches, a sender first obtains the permission from the destination in order to send data packets to it. The destination provides the capabilities to the sender if it wants to receive packets from the sender. The sender then embeds the obtained capabilities into packets. Routers along the path from the sender to the destination verify the capabilities in order to check whether or not the destination wants to receive the packets. If not, the routers simply discard the packets. D-PID differentiates from the capability-based approaches in two aspects. On one hand, communications are initiated by receivers in D-PID but by senders in capability-based approaches. On the other hand, as pointed out in [43], the capability-based approaches are vulnerable to "denial-of-capability" attacks, where compromised computer(s) sends plenty of capability requests to a victim, thus preventing normal users to obtain the capability from the victim. By contrast, D-PID effectively mitigates such attacks because of three reasons. First, the GET messages carry the $PIDs$ along the paths from the compromised computers to the victim. Second, the $PIDs$ are negotiated by neighboring domains that can verify the authenticity of $PIDs$ when they forward GET messages. These two reasons makes it convenient to trace back the attackers. Third, the ubiquitous in-network caching in CoLoR reduces the GET messages sent to the target victim.

Named data networking (NDN) [25] is another approach closely related to our work. In NDN, a content consumer sends out an Interest packet when it wants a piece of content. The Interest is routed (by the content name) to the content provider by routers in the Internet. When a router forwards the Interest toward the content provider, it inserts an entry into its pending Interest table (PIT) that stores the content name and the incoming interface of the Interest packet. When the content provider receives the Interest packet, it sends the corresponding Data packet back to the subscriber. The routers then forward the Data packet back to the content consumer according to the PIT entries stored by them. Unfortunately, maintaining a PIT table at routers makes NDN vulnerable to Interest flooding attacks [44]. By contrast, routers in D-PID do not maintain any forwarding state. In addition, as stated in the previous paragraph, carrying $PIDs$ along the path from attackers to the victim makes it convenient to trace back the attackers, thus help preventing them from launching attacks by sending plenty of GET messages.

VII. Conclusions

In this paper, we have presented the design, implementation and evaluation of D-PID, a framework that dynamically changes path identifiers ($PIDs$) of inter-domain paths in order to prevent DDoS flooding attacks, when $PIDs$ are used as inter-domain routing objects. We have described the design details of D-PID and implemented it in a 42-node prototype to verify its feasibility and effectiveness. We have presented numerical results from running experiments on the prototype. The results show that the time spent in negotiating and distributing $PIDs$ are quite small (in the order of ms) and D-PID is effective in preventing DDoS attacks. We have also conducted extensive simulations to evaluate the cost in launching DDoS attacks in D-PID and the overheads caused by D-PID. The results show that D-PID significantly increases the cost in launching DDoS attacks while incurs little overheads, since the extra number of GET messages is trivial (only 1.4% or 2.2%) when the retransmission period is 300 seconds, and the PID update rate is significantly less than the update rate of IP prefixes in the current Internet.

To the best of our knowledge, this work is the first step toward using dynamic $PIDs$ to defend against DDoS flooding attacks. We hope it will stimulate more researches in this area.

Acknowledgment

We sincerely appreciate the associate editor and the anonymous reviewers for their insightful comments that significantly improve the paper.

References

[1] J. Francois, I. Aib, and R. Boutaba, "Firecol: a Collaborative Protection Network for the Detection of Flooding ddos Attacks," *IEEE/ACM Trans. on Netw.*, vol. 20, no. 6, Dec. 2012, pp. 1828-1841.

[2] OVH hosting suffers 1Tbps DDoS attack: largest Internet has ever seen. [Online] Available: https: //www.hackread.com/ovh-hosting-suffers-1tbps- ddos-attack/.

[3] 602 Gbps! This May Have Been the Largest DDoS Attack in History. http://thehackernews.com/2016/01/biggest-ddos-attack.html.

[4] S. T. Zargar, J. Joshi, D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE Commun. Surv. & Tut.*, vol. 15, no. 4, pp. 2046 - 2069, Nov. 2013.

[5] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks that Employ IP Source Address Spoofing," *IETF Internet RFC 2827*, May 2000.

[6] K. Park and H. Lee, "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," In *Proc. SIGCOMM'01*, Aug. 2001, San Diego, CA, USA.

[7] A. Yaar, A. Perrig, D. Song, "StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense," *IEEE J. on Sel. Areas in Commun.*, vol. 24, no. 10, pp. 1853 - 1863, Oct. 2006.

[8] H. Wang, C. Jin, K. G. Shin, "Defense Against Spoofed IP Traffic Using Hop-Count Filtering," *IEEE/ACM Trans. on Netw.*, vol. 15, no. 1, pp. 40 - 53, Feb. 2007.

[9] Z. Duan, X. Yuan, J. Chandrashekar, "Controlling IP Spoofing through Interdomain Packet Filters," *IEEE Trans. on Depend. and Secure Computing*, vol. 5, no. 1, pp. 22 - 36, Feb. 2008.

[10] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback," In *Proc. SIGCOMM'00*, Aug. 2000, Stockholm, Sweden.

[11] A. C. Snoeren, C. Partridge, L. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-Based IP Traceback," In *Proc. SIGCOMM'01*, Aug. 2001, San Diego, CA, USA.

[12] M. Sung, J. Xu, "IP traceback-based intelligent packet filtering: a novel technique for defending against Internet DDoS attacks," *IEEE Trans. on Parall. and Distr. Sys.*, vol. 14, no. 9, pp. 861 - 872, Sep. 2003.

[13] M. Sung, J. Xu, J. Li, L. Li, "Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Information-Theoretic Foundation," *IEEE/ACM Trans. on Netw.*, vol. 16, no. 6, pp. 1253 - 1266, Dec. 2008.

[14] Y. Xiang, K. Li, W. Zhou, "Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics," *IEEE Trans. on Inf. Foren. and Sec.*, vol. 6, no. 2, pp. 426 - 437, May 2011.

[15] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, S. Shenker, "Off by default!," In *Proc. HotNets-IV*, Nov. 2005, College Park, MD, USA.

[16] A. Yaar, A. Perrig, and D. Song, "SIFF: a stateless internet flow filter to mitigate DDoS flooding attacks," In *Proc. IEEE Symposium on Security and Privacy*, May 2004, Oakland, CA, USA.

[17] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," In *Proc. SIGCOMM'07*, Aug.2007, Kyoto, Japan.

[18] X. Yang, D. Wetherall, and T. Anderson, "TVA: A DoS-Limiting Network Architecture," *IEEE/ACM Trans. on Netw.*, vol. 16, no. 3, pp. 1267 - 1280, Jun. 2008.

[19] X. Liu, X. Yang, and Y. Lu, “To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets,” In *Proc. SIGCOMM'08*, Aug. 2008, Seattle, WA, USA.
[20] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, “Accountable Internet Protocol (AIP),” In *Proc. SIGCOMM'08*, Aug. 2008, Seattle, WA, USA.
[21] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, “Pathlet routing,” in *Proc. SIGCOMM'09*, Aug. 2009, Barcelona, Spain, pp. 111 - 122.
[22] T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKwoen, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, D. Kuptsov, “Architecting for innovation,” *ACM Comput. Commun. Rev.*, vol. 41, no. 3, July 2011, pp. 24 - 36.
[23] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, P. Nikander, “LIPSIN: Line Speed Publish/Subscribe Inter- networking,” in *Proc. SIGCOMM'09*, Aug. 2009, Barcelona, Spain, pp. 195 - 206.
[24] H. Luo, Z. Chen, J. Cui, H. Zhang, M. Zukerman, C. Qiao, “CoLoR: an information-centric internet architecture for innovations,” *IEEE Network*, vol. 28, no. 3, pp. 4 - 10, May 2014.
[25] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” *ACM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66 - 73, Jul. 2014.
[26] T. Koponen, M. Chawla, B. C G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, I. Stoica, “A data-oriented (and beyond) network architecture,” in *Proc. SIGCOMM'07*, Aug. 2007, Kyoto, Japan, pp. 181 - 192.
[27] D. Raychaudhuri, K. Nagaraja, A. Venkataramani, “MobilityFirst: a robust and trustworthy mobility-centric architecture for the future Internet,” *Mobile Comput. and Comm. Rev.*, vol. 16, no. 3, pp. 2 - 13, Jul. 2012.
[28] M. Antikainen, T. Aura, M. Sarela, “Denial-of-service attacks in bloom-filter-based forwarding,” *IEEE/ACM Trans. on Netw.*, vol. 22, no. 5, pp. 1463 - 1476, Oct. 2014.
[29] H. Luo, Z. Chen, J. Cui, H. Zhang, “An Approach for Efficient, Accurate, and Timely Estimation of Traffic Matrices,” In *Proc. IEEE Global Internet Symposium (GI'14)*, May 2014, Toronto, Canada, pp. 67-72.
[30] H. Luo, J. Cui, Z. Chen, M. Jin, H. Zhang, “Efficient integration of software defined networking and information-centric networking with CoLoR,” in *Proc. IEEE GLOBECOM'14*, Dec. 2014, Austin, TX, USA, pp. 1962-1967.
[31] Z. Chen, H. Luo, J. Cui, M. Jin, “Security analysis of a future Internet architecture,” in *Proc. IEEE ICNP'13*, Oct. 2013, Gottingen, Germany.
[32] The BGP instability report. http://bgpupdates.potaroo.net/instability/bgpupd.html/.
[33] BGP Peer Report. http://bgp.he.net/report/peers/.
[34] CAIDA Ark IPv4 Routed /24 AS Links Dataset. http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml/.
[35] H. Jiang, and C. Dovrolis, “Passive estimation of TCP round-trip times,” *ACM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 75 - 88, 2002.
[36] Data Plane Development Kit.http://www.dpdk.eu/.
[37] Click Router. http://www.read.cs.ucla.edu/click/.
[38] Internet AS-level Topology Archive. http://irl.cs.ucla.edu/topology/.
[39] R. Oliveira, B. Zhang, and L. Zhang, “Observing the evolution of Internet AS topology,” *In Proc. SIGCOMM'07*, Aug. 2007, Kyoto, Japan, pp. 313 - 324.
[40] Z. M. Mao, V. Sekar, O. Spatscheck, J. V. D. Merwe, R. Vasudevan, “Analyzing large DDoS attacks using multiple data sources,” In Proc. *SIGCOMM workshop on Large-scale attack defense*, Sep. 2006, Pisa, Italy, pp. 161 - 168.
[41] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” In *Proc. SIGCOMM IMC'10*, Nov. 2010, Melbourne, Australia, pp. 267 - 280.
[42] The CAIDA UCSD Anonymized Internet Traces. http://www.caida.org/data/passive/.
[43] K. Argyraki and D. R. Cheriton, “Network Capabilities: The Good, the Bad and the Ugly,” in *Proc. ACM HotNets'05*, Nov. 2015, College Park, Maryland, USA.
[44] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, “DoS&DDoS in named-data networking,” in *Proc. IEEE ICCCN'13*, Aug. 2013, Nassau, Bahamas.
[45] Z. Chen, H. Luo, M. Zhang, J. Li, “Improving network security by dynamically changing path identifiers in future Internet,” in *Proc. IEEE GLOBECOM'15*, Dec. 2015, San Diego, CA, USA.

Hongbin Luo received the B.S. degree from Beihang University in 1999 and the M.S. (with honors) and Ph.D. degrees in communications and information science from University of Electronic Science and Technology of China (UESTC), in June 2004 and March 2007, respectively.

He is now a professor at the School of Computer Science and Engineering, Beihang University. From June 2007 to March 2017, he worked at the School of Electronic and Information Engineering, Beijing Jiaotong University. From September 2009 to September 2010, he was a Visiting Scholar at the Department of Computer Science, Purdue University. He has authored more than 50 peer-reviewed papers in leading journals (such as IEEE/ACM Transactions on Networking, IEEE Journal on Selected Areas in Communications) and conference proceedings. In 2014, he won the National Science Fund for Excellent Young Scholars from the National Natural Science Foundation of China (NSFC). His research interests are in the wide areas of network technologies including routing, Internet architecture, and optical networking.

Zhe Chen received his B.S. and Ph.D. degrees in communication and information systems from the Beijing Jiaotong University, Beijing, China, in 2011, and 2016, respectively.

He is now with Huawei Technologies. His research interests include future Internet architecture and routing protocols.

Jiawei Li received the B.E. degree in communication and information systems from the Beijing Jiaotong University, Beijing, China, in 2014. He is currently pursuing the Ph.D. degree at the School of Electronic and Information Engineering, Beijing Jiaotong University. He has participated in two National Basic Research Programs of China (“973 Program”), including the Smart Identifier Networks which aims at developing a clean-slate Future Internet architecture.

Athanasios V. Vasilakos is recently Professor with the Lulea University of Technology, Sweden. He served or is serving as an Editor for many technical journals, such as the IEEE Transactions on Network and Service Management; IEEE Transactions on Cloud Computing, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Cybernetics; IEEE Transactions on Nanobioscience; IEEE Transactions on Information Technology in Biomedicine; ACM Transactions on Autonomous and Adaptive Systems; the IEEE Journal on Selected Areas in Communications. He is also General Chair of the European Alliances for Innovation (www.eai.eu).