Inferring the AS-level Topology of Internet with Path Identifiers Used in Inter-Domain Routing

Hongbin Luo, *Member, IEEE,* Hongyi Li, Shan Zhang, *Member, IEEE,* Zhiyuan Wang, Yuxin Mao

***Abstract*—As a promising clean-slate research on the future Internet architecture, CoLoR is information-centric and characterized by its potential benefits such as intrinsic security and evolvability, achieved by decoupling the inter-domain routing from intra-domain routing and using cryptographic path identifiers (*PIDs*) as inter-domain routing objects. In this paper, we propose to infer the AS-level topology of the future Internet architecture under CoLoR. Results from extensive simulations demonstrate that the proposed approach can infer the AS-level topology with an accuracy higher than 95%. The inferred AS-level topology can be used to monitor network traffic, detect traffic anomaly and traceback attackers, thus is helpful for enhancing network security and facilitating network management. While CoLoR is still in its infancy, this work is an important step towards the understanding and exploring new Internet architecture.**

***Index Terms*—Future Internet, Internet topology, path identifiers, inference.**

I. Introduction

ALTHOUGH the current Internet has made tremendous success over the past decades, it still faces some architectural flaws (e.g., [1], [2]). The most frequently cited one is that "*the lack of a coherent security design that has left the Internet vulnerable to various attacks such as DDoS and route spoofing*" [3]. The second widely-accepted one is that "*the current Internet architecture is firmly entrenched, so attempts to significantly change it are unlikely to succeed*" [3]. In addition, it is a wide consensus that the point-to-point packet delivery model of the Internet is not suitable for the current usage reality, where users care about the content more than its location [4]. As a result, many "clean slate" redesigns are proposed, ranging from the new security solutions (e.g., [5], [6]) to more evolvable architectures (e.g., [3], [7], [8]) to information-centric networking (e.g., [9]–[11]).

Among these efforts, CoLoR is a promising *information-centric* Internet architecture characterized by its unique synthesis of four ideas (whose details are described in Section II) [12]. First, CoLoR decouples inter-domain routing from intra-domain routing so that each autonomous system (AS, or domain) can freely deploy its preferred network architecture based on its network environment and preference. Second, CoLoR assigns a block of continuous path identifiers (*PIDs*) that share the same prefix to inter-domain (either physical or logical) paths, and uses PID-prefixes as the inter-domain routing hints to forward packets across domains. Third, CoLoR assigns names to contents and propagates the reachability of content names across domains, as users are more interested in contents and the role of Internet is to distribute content from content providers to content consumers. Last but perhaps the most important, CoLoR associates the service location with the inter-domain routing while decoupling them from forwarding. Specifically, when a request from a content consumer is forwarded from an *AS A* to a neighbor *AS B*, the request is appended with an N-bit cryptographic *PID* that is composed of 1) an n-bit PID-prefix of the inter-domain path connecting *AS A* and *AS B*, 2) an $(N-n)$-bit cryptographic hash over some information carried in the request and a periodically changing secret number known only by routers in $AS\ A$. The *PIDs* carried by the request are then used by the content provider to send the desired content to the content consumer along the inter-domain paths indicated by the *PIDs*.

Due to its unique synthesis of the above four ideas, CoLoR has the potential to provide many important benefits. First, since an AS can deploy a novel network architecture without global coordination, CoLoR allows multiple architectures to coexist and provides intrinsic evolvability. Second, the use of cryptographic *PIDs* as inter-domain routing objects makes it easy for a border router to check whether or not a packet is desired by a customer. This in turn makes it possible to build intrinsic security into CoLoR since data packets can be forwarded to a customer only when a content provider receives a request from the customer [13]–[15]. Third, using *PIDs*-prefixes as inter-domain routing hints significantly reduces the inter-domain routing table size from the current 865K [16] to about 30K because an AS only has limited number of inter-domain paths [17]. In addition, carrying *PIDs* in data packets makes it easy to timely, accurately, and efficiently measure the traffic matrices of a network [18], and facilitates efficient integration of information-centric networking and software-defined networking [19]. Similarly, CoLoR's information-centric nature advocates in-network caching, which is beneficial for reducing the content retrieval delay, alleviating network congestion and saving energy [20]. Furthermore, the information-centric nature and the use of *PIDs* makes it quite efficient to control incoming inter-domain traffic [21], [22].

Despite the potential, CoLoR is still in its infancy and many important issues still need to be addressed. As a step forward, this paper investigates **how to infer the AS-level topology based on the PID sequences under the CoLoR architecture**. Let us elaborate this problem with the example

Hongbin Luo and Shan Zhang are with the School of Computer Science and Engineering, Beihang University, Beijing 100191, China. They are also with the Beijing Key Lab for Novel Networking Technologies (Email: {luohb, zhangshan18}@buaa.edu.cn).

Hongyi Li is with Beijing Research Institute, China Telecom, Beijing 102200, China. (Email: lihy1.bri@chinatelecom.cn).

Zhiyuan Wang and Yuxin Mao are with the School of Computer Science and Engineering, Beihang University, Beijing 100191, China. (Email: {zhiyuanwang,yx990119}@buaa.edu.cn).

in Fig. 1. If the customer C_1 sends a request to an observer O in AS_6, the request will be forwarded along the AS-path composed by inter-domain paths P_1, P_3, and P_6. The observer O then collects the *PID* sequence (i.e., 2.105.3.9, 1.6.13.32, 4.34.36.12) accordingly. If customers in AS_1, AS_2, and AS_4 send many requests to the observer O, then the observer O can collect a set of *PID* sequences and infer the AS-level topology using the method proposed in this paper.

A. Motivation and Challenges

Motivation: The knowledge on the AS-level topology brings many benefits to CoLoR. First, with the AS-level topology, a content provider is able to traceback where a request comes from by matching the *PIDs* carried in the request with the topology, to protect against interest flooding attacks (where a malicious user requests a content persistently). Second, based on the origin *ASs* of requests, a content provider can monitor the number of requests originated from each *AS* to predict its traffic volume from each *AS* and identify traffic anomaly. In addition, a content consumer is able to traceback where a content provider locates, thus preventing a malicious content provider from launching attacks by using collected legitimate *PIDs*. Furthermore, a border router can traceback which *AS* a data packet comes from and which *AS* it is destined to. This again makes it possible to monitor traffic volume between two *ASes* and detect traffic anomaly, thus enhancing the network security.

Challenges: It is non-trivial to obtain the AS-level topology under the CoLoR architecture. The classic approaches (e.g., [23], [24]) proposed to infer the AS-level topology of the current Internet are not applicable to CoLoR, since the current Internet does not use *PIDs* for inter-domain routing. Roughly speaking, inferring the AS-level topology in CoLoR needs to address at least four challenges. First, the observer is unaware of the PID-prefixes assigned to inter-domain paths, because PID-prefixes are not globally advertised. Second, to enhance the security, PID-prefixes may have different lengths. For example, 1.0.0.0/9 assigned to P_7 in Fig. 1 has a prefix length of 9 bits but 1.0.0.0/8 has a prefix length of 8 bits. Third, the requests generating the set of *PID* sequences come from many *ASes* along different AS-paths (e.g., $P_1-P_3-P_6$, $P_2-P_3-P_6$, P_5), which are unknown to the observer. Finally, PID-prefixes may be overlapped and are not globally unique. In Fig. 1, *PIDs* belonging to the PID-prefix 1.0.0.0/9 also belong to the PID-prefix 1.0.0.0/8 assigned to both P_3 and P_5.

B. Main Contributions

In this paper, we propose an approach to infer the AS-level topology under CoLoR, relying on the following features:

1) Each bit from a cryptographic hash is random and evenly distributed (0 or 1, each with a probability 0.5).
2) The PID-prefixes assigned to inter-domain paths connecting an *AS* to other *ASes* are locally unique and do not have overlapping.
3) The sequence of *PIDs* carried by a request has an order, with the outermost one (e.g., 4.34.36.12) corresponding to the inter-domain path (e.g., P_6) closest to the observer and the innermost one (e.g., 2.105.3.9) corresponding to the farthermost inter-domain path (e.g., P_1).
4) For the *PID* sequences generated by requests sent out by the same *AS* and forwarded to the observer along the same AS-path, they must have the same number (denoted by H) of *PIDs*.

To evaluate the performance of the proposed approach, we conduct extensive simulations over four Internet-like topologies generated with the method in [26], which have 3,000, 5,000, 10,000 and 20,000 *ASes*, respectively. The results show that our approach can correctly infer more than 95% PID-prefixes and inter-domain paths.

C. Organization of the Paper

The rest of this paper is structured as follows. Section II provides an overview on CoLoR and its cryptographic *PIDs*. Section III provides an overview on our proposed method. Section IV introduces how to infer possible PID-prefixes from a set of *PIDs*. Based on this, we present how to infer an initial AS-level topology and how to correct errors of the initial topology in Sections V and VI, respectively. Section VII presents simulation results. Finally, we conclude the paper in Section VIII.

II. CoLoR and Its Use of Cryptographic *PIDs*

In this section, we provide an overview on CoLoR to lay the foundation of this work. As the Internet today, CoLoR assumes that a future Internet will still be composed by autonomous systems (*ASes*, or domains). Rather than using IP addresses for both inter-domain routing and intra-domain routing, CoLoR decouples inter-domain routing from intra-domain routing so that each *AS* can deploy its preferred network architecture based on its network environment and preference. As such, an *AS* can choose to deploy a novel network architecture without global coordination, as long as it reaps some benefits. This way, CoLoR provides intrinsic evolvability and allows multiple architectures to coexist.

CoLoR uses PID-prefixes as inter-domain routing objects. Particularly, each inter-domain path is assigned a block of continuous N-bit *PIDs* whose leftmost n ($< N$) bits are the same and are defined as the PID-prefix with length n. In order to correctly forward packets, the PID-prefix assigned to an inter-domain path P connecting two *ASes* is locally unique in the sense that it is not assigned to any other inter-domain paths connecting the two *ASes* to other *ASes*. In addition, the PID-prefixes assigned to inter-domain paths connecting the same *AS* should not be overlapped (i.e., no common *PIDs*). Furthermore, for privacy considerations, a PID-prefix assigned to an inter-domain path is only advertised to routers in the two *ASes* that the path connects, because *ASes* want to keep their interconnections confidential. For example, since PID-prefix 1.0.0.0/8 is assigned to inter-domain path P_3 in Fig. 1, it cannot be assigned to paths P_1 and P_2 that connect AS_3 to AS_1 and AS_2, respectively. Similarly, it cannot be assigned to paths P_4 and P_6. However, PID-prefixes assigned to inter-domain paths could be duplicated, as long as the locally unique property is guaranteed. For instance, PID-prefix 1.0.0.0/8 can

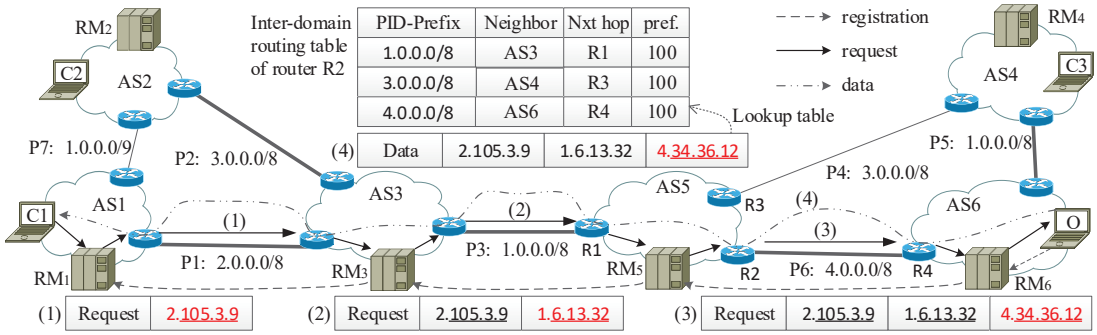


Fig. 1. Illustration for CoLoR and its use of cryptographic *PIDs*.

also be assigned to inter-domain path P_5 connecting AS_4 and AS_6. In addition, PID-prefixes may not have the same length because an AS with very limited number of neighbors can assign PID-prefixes with shorter lengths (thus longer cryptographic hashes) to improve security.

Since *ASes* may use different network architectures and PID-prefixes are not globally unique and advertised, CoLoR assigns names to contents and advertises the reachability of content names. Particularly, as in [9], each *AS* maintains a logically centralized resource manager (RM) and a content provider advertises the names of its hosted contents to its local RM, which then propagates the content names to their neighboring *ASes*, as shown by the dashed lines in Fig. 1. When a consumer wants to obtain a piece of content, it sends out a request carrying the name of the content to its local RM. If a local node hosting the content is found, the RM forwards the request to that node; otherwise, the RM forwards the request to the RM in a neighboring *AS*. Before forwarding the request, the RM computes a cryptographic *PID* for the request and appends the *PID* at the end of the request. In Fig. 1, before the request from C_1 is transmitted from AS_1 to AS_3, RM_1 appends *PID* 2.105.3.9 at the end of the request, as shown by (1), where 2 is the 8-bit PID-prefix and 105.3.9 is the 24-bit hash. Similarly, when the request is forwarded from AS_3 to AS_5, it is appended with the *PID* 1.6.13.32, as illustrated by (2). When it is forwarded from AS_5 to AS_6, *PID* 4.34.36.12 is further appended, as illustrated by (3). Accordingly, when the source O in AS_6 receives the request, it obtains a sequence of *PIDs* and sends the desired content through data packets by using the obtained *PIDs*.

To forward data packets, a border router in an *AS A* maintains an inter-domain routing table that records, for each inter-domain path P connecting *AS A* and its neighbor *AS B*, the PID-prefix assigned to P, the neighbor *AS B*, the next hop router to reach neighbor *AS B*, the preference to path P, as illustrated in Fig. 1. When receiving a data packet carrying a *PID*, as shown by (4), the border router finds out an entry in the inter-domain routing table by using a longest prefix matching algorithm. In addition, the border router checks whether the *PID* is valid by using the same method that the *PID* is generated. If the *PID* is valid, the border router forwards the data packet toward the customer. Otherwise, it discards the data packet. This way, the customer can prevent from receiving data packets from unwanted content sources, thus improving the network security. While an attacker can send requests to attack a content source, the content source can use the *PIDs* carried in requests to traceback the attacker with the help of the inferred AS-level Internet topology, using the approach proposed in this work.

III. Overview on Our Proposed Approach

For presentation convenience, this section provides a brief overview on our proposed approach. Roughly speaking, our proposed approach consists of three steps, which are presented in Section IV, Section V, and Section VI, respectively.

Step 1: Infer PID-Prefixes from a set of PIDs according to Algorithm 1 in Section IV.
Step 2: Infer AS-level topology according to the following procedure in Section V.

1) Infer AS-level skeleton using PID sequences of the largest number of PIDs in Section V-B.
   - First, infer the AS paths and their PID-prefixes according to Algorithm 2.
   - Second, merge the AS paths to obtain the skeleton according to Algorithm 3.
2) Expand the AS-level skeleton using a divide-and-conquer method in Section V-C.
   - First, find *PID* sequences that are either partially covered or not covered by the skeleton (to be defined in Section V-A) and truncate them into truncated *PID* sequences according to Algorithm 4
   - Second, generate a subgraph from each subset of *PID* sequences, and combine it into the skeleton according to Algorithm 5

Steps 3: Correct errors to obtain the final AS-level topology in Section VI.

1) Correct errors caused by wrong aggregation according to Algorithm 7 in Section VI-B.
2) Correct error caused by prefix conflict according to Algorithm 8 according to Section VI-C.

IV. Inferring PID-prefixes From a Set of *PIDs*

In this section, we present an algorithm to infer possible PID-prefixes from a set $\mathcal{S}$ of *PIDs*. Our proposed method

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
PID_0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	0	1	0	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1
$lcp(PID_0, PID_1) = 5$								$lcp(PID_0, PID_2) = 6$						$lcp(PID_0, PID_3) = 7$						$lcp(PID_0, PID_4) = 8$						$lcp(PID_0, PID_5) = 9$								
PID_1	0	0	0	0	0	**1**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PID_2	0	0	0	0	0	0	**1**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PID_3	0	0	0	0	0	0	0	**0**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PID_4	0	0	0	0	0	0	0	1	**1**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PID_5	0	0	0	0	0	0	0	1	0	**1**	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Fig. 2. An example for explaining the divisibility of a set.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	231	147	69	31	11	6	3	1	0	0	1	0	0	0	0	0	1

Fig. 3. A counter vector of the PID set $\mathcal{S}$ that contains $|\mathcal{S}| = 500$ PIDs and share the same 16-bit PID-prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	612	0	0	518	0	0	0	0	0	0	231	147	69	31	11	6	3	1	0	0	1	0	0	0	0	0	1

Fig. 4. A counter vector of the PID set $\mathcal{S}'$ that is generated by adding extra PIDs to set $\mathcal{S}$.

is based on the statistical characteristics of the PIDs sharing the same PID-prefix. Hence let us start with introducing these characteristics in Section IV-A and Section IV-B based on the counter vector and zero hole, respectively. We then present the algorithm in Section IV-C.

A. Counter Vector

Given a PID_0 and any other PID_j in the set $\mathcal{S}$, one could compute their longest common prefix using a longest prefix matching algorithm. Let $lcp(PID_0, PID_j)$ denote the longest common prefix of PID_0 and PID_j. Obviously, the length of any longest common prefix is no greater than 32. Fig. 2 provides some illustrative examples with respect to PID_0. We define the *counter vector* based on the longest common prefix as follows:

Definition 1 (Counter Vector)**.** *For any PID_0 in the set $\mathcal{S}$, we define a series of subsets $\{\mathcal{S}_m^{PID_0} : 0 \le m \le 32\}$ as follows:*

$$\mathcal{S}_m^{PID_0} \triangleq \big\{PID \in \mathcal{S} : \ lcp(PID_0, PID) = m\big\}. \tag{1}$$

The counter vector $CV(PID_0, \mathcal{S})$ is defined as

$$CV(PID_0, \mathcal{S}) \triangleq \big\{\big|\mathcal{S}_m^{PID_0}\big| : \forall m = 0, 1, ...32\big\}, \tag{2}$$

where $|\cdot|$ represents the number of elements in the set.

According to Definition 1, the m-th element of the counter vector $CV(PID_0, \mathcal{S})$ represents the number of *PIDs* that share the same first m bits with PID_0 in the set $\mathcal{S}$. For notation simplicity, we sometimes use $\{C[m] : m = 0, 1, ..., 32\}$ to represent a generic counter vector $CV(PID_0, \mathcal{S})$.

The concept of counter vector is very crucial. Because the PIDs sharing the same PID-prefix exhibit some unique characteristics, which are reflected in the corresponding counter vector. Now let us elaborate these characteristics. Specifically, if a set $\mathcal{S}$ of *PIDs* share the same n-bit PID-prefix, then for any $PID \in \mathcal{S}$, the right $(32 - n)$ bits are independent and identically distributed random variables, which follow Bernoulli distribution with $p = 0.5$. That is, each of these right $(32-n)$ bits takes the values $\{0, 1\}$ with the equal probability and independently with others. Based on this observation, we present the statistical characteristics of the counter vector $CV(PID_0, \mathcal{S})$ in Proposition 1.

Proposition 1. *Suppose that a set $\mathcal{S}$ of PIDs sharing the same n-bit PID-prefix. For a generic $PID_0 \in \mathcal{S}$, the counter vector $CV(PID_0, \mathcal{S})$ satisfies*

$$C[m] = 0, \qquad \text{if } m \in \{0, 1, ..., n-1\}, \tag{3a}$$

$$C[m] \approxeq \frac{|\mathcal{S}|}{2^{m-n+1}}, \qquad \text{if } m \in \{n, ..., 31\}, \tag{3b}$$

$$C[m] \approxeq \left(1 - \sum_{i=n}^{31} \frac{1}{2^i}\right) |\mathcal{S}|, \qquad \text{if } m = 32, \tag{3c}$$

where $\approxeq$ represents "statistically equal" in this paper.

Proof of Proposition 1. We prove the above three equality relations (3) in the following two steps.

First, when the length of the PID-prefix in the set $\mathcal{S}$ is n, the longest common prefix $lcp(PID_0, PID_j)$ is no less than n for any $PID_j \in \mathcal{S}$. This yields (3a).

Second, the length of the PID-prefix is n, thus the $(n+1)$-th bit takes values $\{0, 1\}$ with the equal probability (i.e., feature 1 mentioned in Section I-B). In this case, we have $C[n] \approxeq \sum_{i=n+1}^{32} C[i] \approxeq |\mathcal{S}|/2$. Similar intuition also holds for the $(n+2)$-th bit. This yields (3b) and (3c). □

Now we use the two examples in Fig. 3 and Fig. 4 to illustrate Proposition 1.

- Fig. 3 shows an example for counter vector $CV(PID_0, \mathcal{S})$, where PID_0 is defined in the second line of Fig. 2, and the set $\mathcal{S}$ contains $|\mathcal{S}| = 500$ PIDs sharing the same 16-bit PID-prefix. Note that $\{C[m] : \forall m = 0, 1, ..., 15\}$ is consistent with (3a). Moreover, $\{C[m] : \forall m = 16, 17, ..., 32\}$ follows the characteristics in Proposition 1 with some small fluctuations.

- Fig. 4 shows the counter vector $CV(PID_0, \mathcal{S}')$. Specifically, we generate the set $\mathcal{S}'$ by adding to $\mathcal{S}$ extra PIDs, such that the PIDs in the set $\mathcal{S}'$ have different PID-prefixes. The first $n = 16$ bits in this counter vector contains evidence that indicates us the set $\mathcal{S}'$ of PIDs have different PID-prefixes. To better represent this evidence, we introduce another crucial concept called "zero hole".

B. Zero Hole

The characteristics of PIDs sharing the same PID-prefix lie in the counter vector in Definition 1. To identify these characteristics, we define zero hole of the counter vector in Definition 2.

Definition 2 (Zero Hole)**.** *Given a generic counter vector* $\{C[m] : m = 0, 1, ..., 32\}$*, we say that there is a "**zero hole**" between* m_l *and* m_r*, denoted by* $ZH(m_l, m_r)$*, if the following conditions hold*

$$\begin{aligned} &C[m_l] \neq 0, \\ &C[m_r] \neq 0, \\ &C[m] = 0, \ \forall m \in \{m_l + 1, ..., m_r - 1\}. \end{aligned} \tag{4}$$

Accordingly, m_l *and* m_r *represent the left and right boundaries of the zero hole* $ZH(m_l, m_r)$*, respectively.*

We use the two examples in Fig. 3 and Fig. 4 to illustrate the above definition. Specifically, there are a total of two zero holes in Fig. 3, i.e., $ZH(23, 26)$, and $ZH(26, 32)$. In Fig. 4, there are a total of four zero holes, i.e., $ZH(6, 9)$, $ZH(9, 16)$, $ZH(23, 26)$, and $ZH(26, 32)$. Proposition 2 introduces why the presence of zero holes may imply that the corresponding set of PIDs have multiple PID-prefixes.

Proposition 2. *Suppose that the set* $\mathcal{S}$ *of PIDs have the same PID-prefix. Given the counter vector* $CV(PID_0, \mathcal{S})$*, a zero hole* $ZH(m_l, m_r)$ *exists with the following probability*

$$\left(\frac{1}{2}\right)^{(m_r - m_l)\sum_{i \geq m_r}^{32} C[i]}. \tag{5}$$

Proof of Proposition 2. If the zero hole $ZH(m_l, m_r)$ appears in the counter vector $CV(PID_0, \mathcal{S})$, then there are at least $\sum_{i \geq m_r}^{32} C[i]$ PIDs in set $\mathcal{S}$ holding the same bits $\{m_l, ..., m_r - 1\}$ as PID_0. Such a phenomenon occurs with the probability (5). This completes the proof of Proposition 2. □

According to Proposition 2, the presence of $ZH(6, 9)$ and $ZH(9, 16)$ in Fig. 4 implies that the set $\mathcal{S}'$ of PIDs have the same PID-prefix with an extremely low probability. However, the zero holes $ZH(23, 26)$ and $ZH(26, 32)$ in Fig. 3 appears with a relatively high probability. This motivates us to focus on the "valid zero hole" defined as follows:

Definition 3 (Valid Zero Hole)**.** *A zero hole* $ZH(m_l, m_r)$ *is said to be "valid" if the following inequality holds*

$$\sum_{i=m_r}^{32} C[i] \geq 9. \tag{6}$$

The inequality (6) is used to exclude those zero holes, e.g., $ZH(23, 26)$ and $ZH(26, 32)$. Here we choose the threshold 9, since it satisfies $2^8 < M = 385 < 2^9$, where M is our minimal PID sample size. Note that the statistical characteristics only hold when the set $\mathcal{S}$ is sufficiently large. One could use the Cochran's formula [25] to derive the minimal sample size given the desired precision. Specifically, given the precision e, the minimal sample size $M(e)$ is given by

$$M(e) \triangleq \frac{u_{\alpha/2}^2}{4e^2}, \tag{7}$$

where α is the desired confidence level and $u_{\alpha/2}$ represents the quantile for $\alpha/2$ of standard normal distribution. In this paper, the confidence level is set to $\alpha = 95\%$ (thus we have $\mu_{\alpha/2} = 1.959$) and the precision is $e = 5\%$. thus we have $M = 385$. We will focus on the *PID* set that is no less than $M = 385$ (i.e., $|\mathcal{S}| \geq M$).

Algorithm 1: Inferring PID-Prefix

Input: A set $\mathcal{S}$ of PIDs.
Output: The PID-prefixes.

Step 1: Select a $PID_0 \in \mathcal{S}$ randomly;
Step 2: Calculate the counter vector $CV(PID_0, \mathcal{S})$
Step 3: Check whether $CV(PID_0, \mathcal{S})$ has valid zero hole.
If yes, go to Step 4.
If no, go to Step 5.
Step 4: Find the minimal m_s such that $C[i] = 0$ for any $i < m_s$, and think the first m_s bits are the PID-prefix.
Step 5: Divide $\mathcal{S}$ into subsets and repeat Algorithm 1 for each subset; Specifically, given a valid zero hole $ZH(m_l, m_r)$, divide the PID set $\mathcal{S}$ into $\mathcal{S}_{m_l}^{PID_0}$ and $\mathcal{S} \backslash \mathcal{S}_{m_l}^{PID_0}$.

C. Inferring PID-Prefix

Based on the above discussion, if a set $\mathcal{S}$ of PIDs have the same PID-prefix, then the corresponding counter vector $CV(PID_0, \mathcal{S})$ should not have valid zero holes. One could infer the PID-prefix using this characteristic. Algorithm 1 summarizes the procedure of inferring the PID-prefix from a set $\mathcal{S}$ of PIDs. Specifically, we first randomly draw a $PID_0 \in \mathcal{S}$, and calculate the counter vector $CV(PID_0, \mathcal{S})$. If there is no valid zero hole, then we think the first m_s bits are the PID-Prefix, where $C[i] = 0$ for any $i < m_s$. If there is valid zero hole $ZH(z_l, z_r)$, we first divide the original $\mathcal{S}$ into subsets $\mathcal{S}_{m_l}^{PID_0}$ and $\mathcal{S} \backslash \mathcal{S}_{m_l}^{PID_0}$, and then run this algorithm for both of them again. Here $\mathcal{S}_m^{PID_0}$ is defined in (1). Taking the counter vector in Fig. 4 as an example, Algorithm 1 eventually divides the set $\mathcal{S}'$ into three subsets: 612 PIDs in set $\mathcal{S}_6^{PID_0}$, 518 PIDs in set $\mathcal{S}_9^{PID_0}$, and the remaining 500 PIDs.

So far, we have introduced how to infer the PID-prefix of a set of PIDs. This method will be used in inferring the AS-level topology in Section V.

V. Inferring an Initial AS-level Topology

In this section, we introduce how to infer the AS-level topology based on the collected PID sequences, where we will use the method in Section IV to obtain the PID-prefixes. We first give an overview on the key rationale in Section V-A. We then present the design details in Section V-B and Section V-C, respectively.

A. Overview of the Proposed Approach

We propose to infer the AS-level topology in two steps. Moreover, we will use the example in Fig. 5 to explain our basic rationale and assume that Fig. 5(d) is the resulting inferred AS-level topology.

Step 1: Inferring AS-Level Skeleton based on the PID sequence with the largest number of PIDs. If a *PID* sequence has a large number of *PIDs*, the request carrying this *PID* sequence passes through a large number of inter-domain paths. Accordingly, the *PID* sequences that have the largest number of *PIDs* should provide the most information about the AS-level topology to the observer. Therefore, our first step is to use the *PID* sequences with the largest number of *PIDs* to obtain some PID-prefixes and the corresponding inter-domain paths to form a skeleton of the AS-level topology using the method in Section V-B. For example, we use the *PID* sequences with three *PIDs* to infer a skeleton shown in Fig. 5(c), where an AS is represented by a small circle, the inter-domain path connecting two *ASes* v_1 and v_2 is represented by a solid line and denoted by $e(v_1, v_2)$, the PID-prefix corresponding to the inter-domain path is shown next to the solid line.

Step 2: Expand the AS-level skeleton using a divide-and-conquer method. With the obtained skeleton, we expand the skeleton to obtain the AS-level topology in a divide-and-conquer manner. To proceed, we divide all *PID* sequences received by the observer into three categories based on the obtained skeleton.

- The first category includes the *PID* sequence that is generated by a request sent from an *AS* in the skeleton.
- The second category includes the *PID* sequence that is generated by a request sent from an *AS* out of the skeleton, but the request passes through at least one inter-domain path in the skeleton.
- The third category includes the *PID* sequence that is generated by a request sent from an *AS* out of the skeleton, and the request does not pass through any inter-domain path in the skeleton.

In Fig. 5(d), the *PID* sequences $\{1.2.3.9, 2.0.1.8\}$ and $\{4.4.5.6, 1.5.7.2, 2.2.4.6\}$ belong to the first category, because they are generated by requests sent from *AS* v_{12} and *AS* v_{13}, respectively. On the other hand, a *PID* sequence generated by a request sent from AS v_{71} belongs to the second category, because the request passes through the inter-domain path $e(v_{11}, O)$. A *PID* sequence generated by a request sent from AS v_{62} belongs to the third category because the request does not pass through any inter-domain path in the skeleton.

For ease of presentation, we say that a *PID* sequence in the first category is "covered" by the skeleton. The *PID* sequence in the second category is "partially covered" by the skeleton. The *PID* sequence in the third category is "not covered" by the skeleton. For a *PID* sequence partially covered by the skeleton and generated by a request from AS A, the first AS in the skeleton that the request passes towards the observer is defined as the "break point" of the *PID* sequence. If a *PID* sequence is not covered by the skeleton, its break point is the observer. For example, for the *PID* sequence $\{3.6.4.9, 2.10.15.8\}$, its break point is *AS* v_{11} in Fig. 5(c). On the other hand, for the *PID* sequence $\{4.2.3.9, 5.0.1.8\}$, its break point is O.

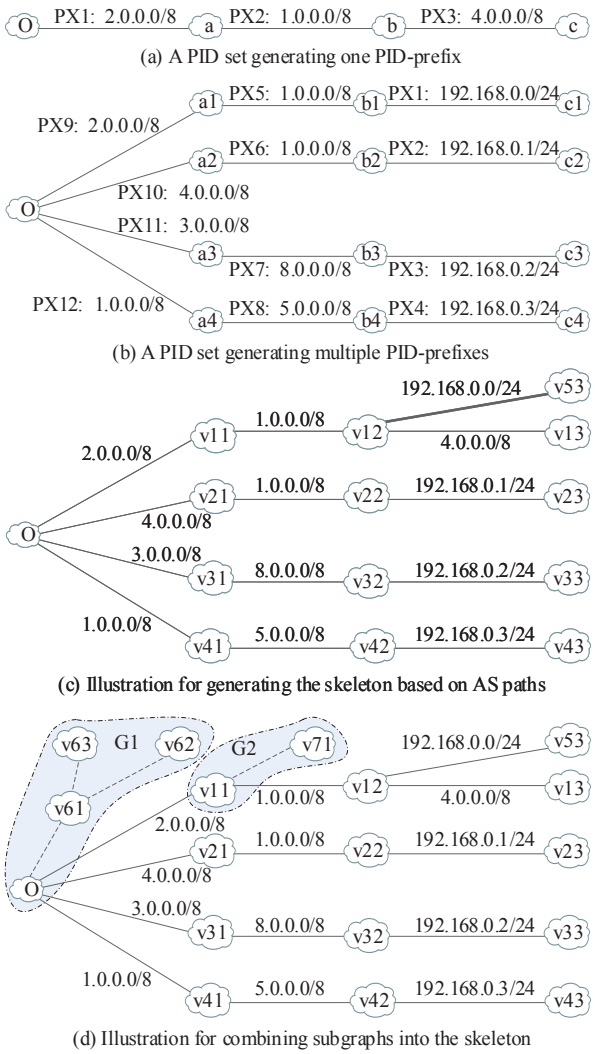


Fig. 5. Illustration for inferring the skeleton of the AS-level topology.

We will expand the skeleton to obtain the AS-level topology in a *divide-and-conquer* method. In particular, based on the break points of the *PID* sequences in the second and third categories, we divide them into multiple subsets such that the *PID* sequences in each subset share the same break point. For the *PID* sequences in each subset, we then infer a subgraph whose root is the common break point of these *PID* sequences. By merging the subgraphs with the skeleton, we then obtain the whole AS-level topology. For example, in Fig. 5(d), we obtain two subgraphs (i.e., G_1 and G_2) from two subsets of *PID* sequences, merge them into the skeleton shown in Fig. 5(c), and obtain the resulting AS-level topology shown in Fig. 5(d). We describe the details in Section V-C.

B. Inferring the Skeleton of the AS-level Topology

We now describe in detail how to infer the skeleton of the AS-level topology by using the *PID* sequences that have the largest number of *PIDs*. Without loss of generality, we use H to denote the number of *PIDs* in a *PID* sequence, and

use $\overline{S}^H$ to denote the set of *PID* sequences with H *PIDs*. Since each *PID* corresponds to an inter-domain path, a *PID* sequence corresponds to the concatenation of H inter-domain paths, which is called as an AS path in the rest of the paper. Accordingly, we infer the skeleton of the AS-level topology in two steps.

- First, we infer the AS paths in the skeleton and their PID-prefixes according to Algorithm 2.
- Second, we merge the AS paths to obtain the skeleton according to Algorithm 3.

Next we introduce the two steps in details.

1) Inferring AS Paths: Given an AS path with H hops, we assume that the hop nearest to the observer is the H-th hop and the one furthermost to the observer is the first hop. For example, for the *PID* sequence {4.1.5.6, 1.0.23.34, 2.0.1.0}, the *PID*s 4.1.5.6 and 2.0.1.0 correspond to the furthermost and the nearest hops, respectively. Accordingly, for every *PID* sequence denoted by $\overline{PID}_i^H$ in the set $\overline{S}^H$, we put the *J-th* ($1 \leq J \leq H$) *PID* into the set S_J = $\{PID_{iJ}\}$ and obtain H sets of *PIDs*. For example, given four *PID* sequences {4.1.5.6, 1.0.23.34, 2.0.1.0}, {4.2.67.52, 1.2.35.6, 2.1.2.4}, {4.69.136.28, 1.56.28.16, 2.12.3.68} and {4.96.137.99, 1.96.126.7, 2.201.125.3}, we obtain three sets of *PIDs* $\mathcal{S}_1$ = {4.1.5.6, 4.2.67.52, 4.69.136.28, 4.96.137.99}, $\mathcal{S}_2$ = {1.0.23.34, 1.2.35.6, 1.56.28.16, 1.96.126.7} and S_3 = {2.0.1.0, 2.1.2.4, 2.12.3.68, 2.201.125.3}. Because the *PID* sequences in the set $\overline{S}^H$ may come along different AS paths, *PIDs* in the set S_J may correspond to multiple PID-prefixes. For example, in Fig. 5(b), *PIDs* in the set $\mathcal{S}_2$ generate three PID-prefixes (i.e., 1.0.0.0/8, 5.0.0.0/8, and 8.0.0.0/8), while *PIDs* in the set $\mathcal{S}_1$ generate two PID-prefixes (i.e., 192.168.0.0/22, 4.0.0.0/8). In this case, the PID-prefix 192.168.0.0./22 is a wrong aggregation of four PID-prefixes (192.168.0.0/24, 192.168.0.1/24, 192.168.0.2/24, and 192.168.0.3/24) that cannot be split by Algorithm 1.

Algorithm 2 first infers the possible PID-prefixes from *PIDs* in the set $\mathcal{S}_1$ and the prefix lengths by using Algorithm 1. Without loss of generality, we assume that we obtain M_{px} PID-prefixes denoted by PX_j ($j = 1, 2, ..., M_{px}$).

- Case of $M_{px} = 1$: If there is only one PID-prefix in the set $\mathcal{S}_1$ (e.g., 4.0.0.0/8 in Fig. 5(a), 192.168.0.0/22 in Fig. 5(b)), we infer the possible PID-prefixes from *PIDs* in the set $\mathcal{S}_2$ and their prefix lengths. If we also obtain one PID-prefix, we infer the possible PID-prefixes from *PIDs* in the set $\mathcal{S}_3$. This process continues until either we obtain one PID-prefix from the set $\mathcal{S}_H$, or we obtain two or more PID-prefixes from a set $\mathcal{S}_J$. In the first case, we obtain a single AS path, as shown in Fig. 5(a).
- Case of $M_{px} \geq 2$: If there are more than one PID-prefix (e.g., 192.168.0.0/22 and 4.0.0.0/8 in Fig. 5(d)), we divide the *PID* sequences in the set $\bar{\mathcal{S}}^H$ into M_{px} subsets $\mathcal{S}_j^H$ ($j = 1, 2, ..., M_{px}$) of *PID* sequences such that one PID-prefix corresponds to one subset $\mathcal{S}_j^H$. In particular, if the first *PID* of a *PID* sequence belongs to PX_j, the *PID* sequence is put into the subset $\mathcal{S}_j^H$. In Fig. 4(d), the *PID* sequences with three *PIDs* can be divided into two subsets, one for *PID* sequences whose first hop *PID* belongs to 192.168.0.0/22, the other for *PID* sequences whose first hop *PID* belongs to 4.0.0.0/8. For each subset of $\mathcal{S}_j^H$ of *PID* sequences, we also use the method described here to infer the possible AS paths. This process repeats until we cannot obtain two or more PID-prefixes from any set of *PIDs*. In this process, we obtain a set of AS paths.

Algorithm 2: Inferring AS paths from a set of *PID* sequences

Input: a set $\{\overline{PID}_i^H\}$ comprised by M_S *PID* sequences; each sequence has H *PIDs*, with PID_{ij} being the *j-th PID* ($1 \leq j \leq H$).
Output: a set of AS paths $\{ASP_j\}$, where ASP_j is the *j-th* AS path; ASP_j = $\{PX_i^j\}$, with $\{PX_i^j\}$ being the *i-th* PID-prefix of ASP_j.

Initialize: let S_i be a set comprised by *PIDs*, initialize S_i be empty; let $\{ASP_j\}$ be empty.

Step 1: for every *PID* sequence $\overline{PID}_i^H$
  obtain the *H-th PID* denoted by PID_{iH} in $\overline{PID}_i^H$;
  put PID_{iH} into set S_H;
Step 2: employ Algorithm 1 over set S_H, obtain a set of PID-prefixes denoted by $\{PX_j\}$ ($1 \leq j \leq M_H$).
Step 3: for every *PID* sequence $\overline{PID}_i^H$ in set $\{\overline{PID}_i^H\}$
  if $PID_{iH} \in PX_j$, put $\overline{PID}_i^H$ into subset PID_j^H;
 form a set $\{PID_j^H\}$ comprised by subsets PID_j^H;
 let $\{ASP_i\}$ be a set of AS paths and initialize it to be empty;
Step 4: if set $\{PID_j^H\}$ is nonempty
  chooses a subset PID_j^H from $\{PID_j^H\}$;
  remove PID_j^H from $\{PID_j^H\}$;
  let ASP be an AS path, initialize it to be empty;
  let the set S_k ($1 \leq k \leq H$) be empty;
  for every *PID* sequence $\overline{PID_i} \in PID_j^H$
   put the *k-th PID* of $\overline{PID_i}$ into set S_k;
  for $k = H$ to 1 step -1
   let the PID-prefix set $\{PX_m\}$ be empty;
   employ Algorithm 1 over S_k, obtain a set $\{PX_m\}$;
   if there is only one PID-prefix in set $\{PX_m\}$
    denote the PID-prefix by PX_k; add PX_k into ASP;
   else (%there may be multiple PID-prefixes)
    for every *PID* sequence $\overline{PID_j^H}$ in set PID_j^H
     if $PID_{jk} \in PX_m$, put $\overline{PID_j^H}$ into subset PID_m^H;
    put subsets PID_m^H into $\{PID_j^H\}$;

Fig. 5 illustrates the process of inferring the three-hop AS paths. Initially, two PID-prefixes (i.e., 192.168.0.0/22 and 4.0.0.0/8) are obtained, and the three-hop *PID* sequences are divided into two subsets. The first one is for the PID-prefix 4.0.0.0/8 and corresponds to the case in Fig. 5(a). The second one is for the PID-prefix 192.168.0.0/22 and corresponds to the case in Fig. 5(b). When the *PID* sequences in the first subset are considered, we obtain one AS path since only one PID-prefix can be inferred from the *PIDs* in each set S_i ($1 \leq i \leq 3$), as illustrated in Fig. 5(a). By contrast, when the *PID* sequences in the second subset are considered, we also obtain one PID-prefix (i.e., 192.168.0.0/22) from the first hop *PIDs*. We then infer the PID-prefixes from the second hop *PIDs* and obtain three PID-prefixes 1.0.0.0/8, 5.0.0.0/8, and 8.0.0.0/8. Accordingly, we further divide the *PID* sequences in the second subset into three sets, the first $\mathcal{S}_1^3$ for 1.0.0.0/8, the second $\mathcal{S}_2^3$ for 5.0.0.0/8, and the third $\mathcal{S}_3^3$ for 8.0.0.0/8. From the *PID* sequences in $\mathcal{S}_2^3$ and $\mathcal{S}_3^3$, we obtain AS paths O-a_3-b_3-c_3 and O-a_4-b_4-c_4, respectively. By contrast, when the *PID* sequences in $\mathcal{S}_1^3$ is considered, we obtain two PID-prefixes (i.e., 2.0.0.0/8, 4.0.0.0/8) from the third hop *PIDs*. As a result, the *PID* sequences in $\mathcal{S}_1^3$ is further divided into two

Algorithm 3: Merge AS Paths to Obtain AS-Level Skeleton

Input: a set of AS paths $\{ASP_j\}$.
Output: a graph $G = (V, E)$, where V is the set of nodes representing the *ASes* and E is the set of edges (or paths) connecting *ASes*; each edge has a PID-prefix with certain prefix length.

Initialize: let $V = \{O\}$; let E be empty;
let ID be an indicator; $K = 0$;

Step 1: choose an AS path $ASP_1 = \{PX_i^1\}$ from set $\{ASP_j\}$;
  for PX_1^1 in ASP_1
    add a node v_{11} onto the graph;
    add an edge (O, v_{11}) connecting nodes O and v_{11} onto G;
    associate PX_1^1 with the edge (O, v_{11});
  for $2 \leq i \leq H$
    add a node v_{1i} onto the graph; $k = i - 1$;
    add an edge (v_{1k}, v_{1i}) connecting nodes v_{1k} and v_{1i} onto G;
    associate PX_i^1 with the edge (v_{1k}, v_{1i});
  remove the AS path ASP_1 from set $\{ASP_j\}$;
Step 2: if the set $\{ASP_j\}$ is not empty
    choose an ASP $ASP_j = \{PX_i^j\}$ from $\{ASP_j\}$; $K = 0$;
    let U be an anchor node and initialize it to be node O;
    for $i = 1$ to H, $ID = 0$;
      for any edge (U, v_{mn}) connecting nodes U and v_{mn}
        denote the PID-prefix for edge (U, v_{mn}) by PX_U^v;
        if PX_i^j is equal to PX_U^v
          $ID = 1$; record node v_{mn}; break;
      if $ID == 0$, then $K = i$; $ASP = ASP_j$;
        remove ASP_j from set $\{ASP_j\}$; go to Step 3;
      else, $U = v_{mn}$;
  else, end;
Step 3: for $i = K$ to H
    add a new node v_{ji} onto the graph G;
    add a new edge (U, v_{ji}) onto the graph G;
    associate PX_i^j in ASP with edge (U, v_{ji}); $U = v_{ji}$;
  $K = 0$; go to Step 2;

subsets, one for the AS path O-a_1-b_1-c_1 and the other for the AS path O-a_2-b_2-c_2.

2) Merging AS Paths to Obtain the Skeleton: With the obtained AS paths with H hops, we are able to infer the skeleton of the AS-level topology by using Algorithm 3. To this end, we put the obtained AS paths into a set denoted by $\{ASP_i\}$ $(i = 1, 2, ..., M_a)$, where ASP_i is the i-th AS path and M_a is the number of obtained AS paths. Note that each AS path ASP_i has H PID-prefixes. Let PX_i^j $(1 \leq j \leq H)$ denote the j-th PID-prefix of AS path ASP_i. To represent the skeleton of the AS-level topology, we generate a graph $G(V, E)$, where V is the set of nodes representing *ASes* and E is the set of edges (i.e., paths) connecting *ASes*. At the beginning, V is initialized to include only a single node O, which represents the AS that the observer locates. We then choose the first AS path ASP_1 from the set $\{ASP_i\}$. For $ASP_1 = \{PX_1^j\}$, we choose the last PID-prefix PX_1^H, add a node v_{11} onto the graph, and add an edge (O, v_{11}). We then choose the second PID-prefix PX_1^{H-1}, add a node v_{12} onto the graph, and add an edge (v_{11}, v_{12}). Similarly, the nodes $v_{13}, v_{14}, \cdots, v_{1H}$ and edges $(v_{1(j-1)}, v_{1j})$ $(3 \leq j \leq H)$ are added to the graph. For example, given the AS path {4.0.0.0/8, 1.0.0.0/8, 2.0.0.0/8}, we generate the graph G represented by the path O-v_{11}-v_{12}-v_{13} shown in Fig. 5(c).

Once the first AS path is added into the graph, we add the rest (if any) AS paths $\{ASP_i\}$ into the graph as follows. In particular, beginning with the second AS path, we sequentially choose the AS paths from $\{ASP_i\}$. For the chosen AS path $ASP_i = \{PX_i^j\}$, we first pick up PX_i^H and compare it with the PID-prefixes corresponding to the edges connecting the node O with other nodes in the graph. If PX_i^H is not equal to any of these PID-prefixes, a new node v_{i1} and an edge (O, v_{i1}) are added into the graph. In addition, other nodes v_{ij} $(2 \leq j \leq H)$ and the corresponding edges are added into the graph. For example, if the second AS path is {192.168.0.1/24, 1.0.0.0/8, 4.0.0.0/8}, nodes v_{21}, v_{22}, v_{23} and edges (O, v_{21}), (v_{21}, v_{22}), (v_{22}, v_{23}) are added into the graph, as shown in Fig. 5(c). Similarly, for the AS paths {192.168.0.2/24, 8.0.0.0/8, 3.0.0.0/8} and {192.168.0.3/24, 5.0.0.0/8, 1.0.0.0/8}, corresponding nodes and edges are added into the graph, as shown in Fig. 5(c).

However, if PX_i^H is equal to a PID-prefix corresponding to an edge (O, v_{m1}) in the graph, it is not necessary to add a new node into the graph because two PID-prefixes are non-overlapping, as described in Section I. In this case, we compare the prefix PX_i^{H-1} in the AS path ASP_i with the PID-prefixes corresponding to edges connecting node v_{m1} to other nodes in the graph. If PX_i^{H-1} is equal to the PID-prefix corresponding to an edge (v_{m1}, v_{n2}) in the graph, we further compare PX_i^{H-2} with the PID-prefixes corresponding to edges connecting node v_{n2} to other nodes in the graph. This process continues until

1) PX_i^h is equal to the PID-prefix corresponding an edge (v_s, v_t).
2) PX_i^{h+1} is not equal to any PID-prefix corresponding the edges connecting node v_t to other nodes in the graph.

In this case, we add a new node v_i^{h+1} into the graph and add an edge (v_t, v_i^{h+1}) into the graph. In addition, for PX_i^j $(h+2 \leq j \leq H)$, we add nodes and corresponding edges into the graph. For example, given the AS path {192.168.0.0/24, 1.0.0.0/8, 2.0.0.0/8}, the PID-prefix 2.0.0.0/8 equals to the PID-prefix corresponding to edge (O, v_{11}). Thus we do not add a new node into the graph, and consider the second PID-prefix 1.0.0.0/8. Since 1.0.0.0/8 equals to the PID-prefix corresponding to edge (v_{11}, v_{12}), we do not add a new node into the graph. When the PID-prefix 192.168.0.0/24 is considered, it is not equal to the PID-prefix 4.0.0.0/8. Thus, a new node v_{53} and an edge (v_{12}, v_{53}) are added into the graph, as shown in Fig. 5(c).

Finally, we obtain the skeleton of the AS-level topology, when all AS paths are added into the graph G.

C. Expanding the Skeleton

We now expand the skeleton to obtain the AS-level topology by using a *divide-and-conquer* method. We use Algorithm 4 to find *PID* sequences that are either partially covered or not covered by the skeleton, and truncate them into truncated *PID* sequences based on their break points. For example, the *PID* sequence {3.6.4.9, 2.10.15.8} with the break point v_{11} shown in Fig. 5(c) can be truncated to {3.6.4.9}. Similarly, the *PID* sequence {4.2.3.9, 5.0.1.8} with the break point O can be truncated to {4.2.3.9, 5.0.1.8}. Note that we do not consider the *PID* sequences covered by the skeleton since they cannot expand the skeleton.

We then divide the truncated *PID* sequences into (possibly) multiple subsets of *PID* sequences such that the *PID* sequences in each subset have the same break point. For ease of presentation, we use a tuple $< v, \{\overline{PID}\} >$ to

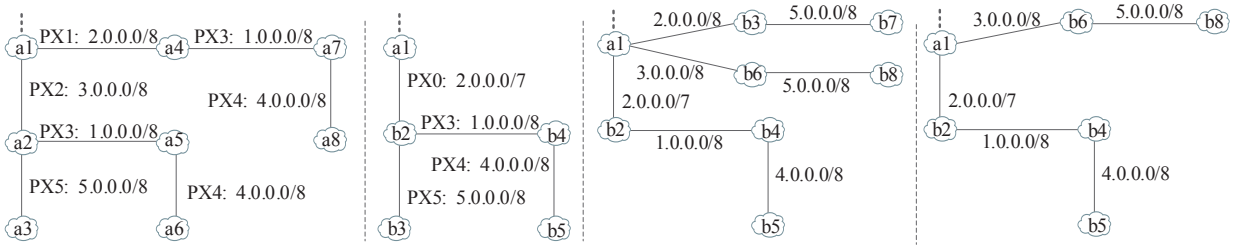


Fig. 6. Illustration for the first type of errors: wrong aggregation.

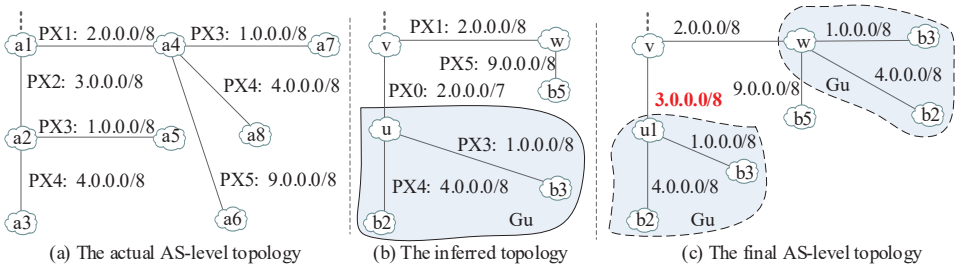


Fig. 7. Illustration for the second type of errors: prefix conflict.

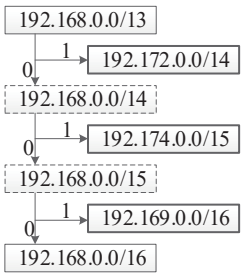


Fig. 8. Splitting a PID-prefix.

Algorithm 4: Finding *PID* sequences that are either partially covered or not covered by a graph
Input: a set of *PID* sequences $\mathcal{S} = \{\overline{PID_i}\}$, a graph $G = (V, E)$, where V is the set of nodes representing the *ASes* and E is the set of edges; each edge has a PID-prefix with certain prefix length. **Output**: a set ST of tuples $\{(v_o, \{\overline{PID_j}\})\}$, where v_o is a break point and $\{\overline{PID_j}\}$ are *PID* sequences with v_o being their break points.
Initialize: Let O be the root node of graph G; for every node v in G, compute its distance l_v to node O, it is apparent that $l_O = 0$.
Step 1: check whether or not the set $\mathcal{S}$ is empty. If so, terminate; otherwise, $n_c = O$, $l = l_O + 1$, go to Step 2. ***Step 2***: choose a *PID* sequence s from $\mathcal{S}$; let $j = l$; remove s from $\mathcal{S}$; let S_{PX} be a set, initialize it to be empty; let $s[j]$ be the *j-th* PID of s. ***Step 3***: for every edge $e(n_c, v)$ with PID-prefix *PX*, put the triple $< v, PX, l_v >$ into set S_{PX} if $l_v > l$; ***Step 4***: if set S_{PX} is empty, go to Step 7; otherwise, go to Step 5. ***Step 5***: check whether or not $s[j]$ belongs to some PID-prefix in set S_{PX}. If so, let $m = j$, record the corresponding triple, go to Step 6; otherwise, go to Step 7. ***Step 6***: if m equals to the length of *PID* sequence s, go to Step 1; otherwise, $n_c = v$, $j = j + 1$, $l = l_v$, where v and l_v are the first and third element of the recorded triple; go to Step 3. ***Step 7***: node n_c is the break point of the *PID* sequence s. delete the first l_v *PIDs* from the *PID* sequence s and obtain a truncated *PID* sequence denoted by s'. check whether or not there is a tuple whose first element is n_c. If so, add s' into the set of *PID* sequences for n_c; otherwise, create a tuple with the first element being n_c,add s' into the set of *PID* sequences for n_c; ***Step 8***: go to Step 1.

denote the *PID* sequences in the same subset, where v is the common break point of the *PID* sequences in set $\{\overline{PID}\}$. For instance, the truncated *PID* sequence $\{3.6.4.9\}$ should be put into the subset $< v_{11}, \{3.6.4.9\} >$; similarly, the truncated *PID* sequence $\{4.2.3.9, 5.0.1.8\}$ should be put into the subset $< O, \{4.2.3.9, 5.0.1.8\} >$.

Based on Algorithm 4, we take each subset of truncated *PID* sequences and their common break point as input, and try to obtain a subgraph G_s from each subset of *PID* sequences by using Algorithms 2, 3, 4 and 5. When a subgraph G_s is obtained, we then combine it into the skeleton. For example, by using the truncated *PID* sequences with the common break point O, we generate the subgraph G_1 in Fig. 5(d), which is a tree with the node O being the root. Similarly, we generate the subgraph G_2 by using the truncated *PID* sequences with the node v_{11} being their common break point. When subgraphs G_1 and G_2 are generated, we combine them into the skeleton and form an AS-level topology in Fig. 5(d). When all subsets are considered, an initial AS-level topology is obtained.

VI. Correcting Errors in the AS-level Topology

When we infer the AS-level topology in Section V, some information is not taken into consideration, which may lead to errors in the obtained AS-level topology. Next we introduce two types of errors in Section VI-A, and our strategies for addressing these issues in Section VI-B and Section VI-C, respectively.

A. Two Types of Errors

Wrong Aggregation: The first error is wrong aggregation. Specifically, the wrong aggregation of PID-prefixes can be resulted when inferring the skeleton, and this error is not corrected in the process of skeleton expansion. Fig. 6(a) shows the PID-prefixes assigned to inter-domain paths. When the longest *PID* sequences are considered, a skeleton with only one AS path $\{4.0.0.0/8,\ 1.0.0.0/8,\ 2.0.0.0/7\}$ is generated because the two PID-prefixes 2.0.0.0/8 and 3.0.0.0/8 are mis-aggregated to be 2.0.0.0/7. When the rest of *PID* sequences are considered in Algorithm 5, they are truncated to include only *PIDs* belonging to the PID-prefix 5.0.0.0/8, because a *PID* (e.g., 3.1.9.8) belonging to the PID-prefix 3.0.0.0/8 is covered by 2.0.0.0/7. As a result, Algorithm 5 obtains a subgraph G_1 that includes only nodes b_2, b_3 and the edge (b_2, b_3). After

Algorithm 5: Expanding the skeleton

Input: a set of *PID* sequences $\mathcal{S} = \{\overline{PID_i}\}$, a graph $G_O = (V, E)$, where V is the set of nodes representing the *ASes* and E is the set of edges; each edge has a PID-prefix with certain prefix length.
Output: a new graph G_n.

Initialize: let G_s be a subgraph, initialize it to be empty; $G_n = G_O$.

Step 1: use Algorithm 4 to divide the *PID* sequences into one or more subsets $\{ST\} = \{< v, \{\overline{PID}\} >\}$;
Step 2: if the set $\{ST\}$ is empty, terminate; else, choose an ST from set $\{ST\}$, obtain the common break point v and the set of *PID* sequences $\{\overline{PID}\}$ from ST; go to Step 3;
Step 3: find the longest *PID* sequences in the set $\{\overline{PID}\}$, put these *PID* sequences into a set S_a;
Step 4: employ Algorithm 2 to infer the possible AS paths;
Step 5: employ Algorithm 3 to form the skeleton of a subgraph G_s whose root node is node v;
Step 6: employ Algorithm 5 to expand the skeleton of subgraph G_s such that all *PID* sequences in set $\{\overline{PID}\}$ are considered;
Step 7: combine subgraph G_s into graph G_n;
Step 8: remove ST from set $\{ST\}$, go to Step 1.

Algorithm 6: Split a shorter PID-prefix into longer PID-prefixes

Input: PID-prefix PX_0^1 with length l_0^1, PID-prefix PX_0^2 with length l_0^2 such that $l_0^1 < l_0^2$;
Output: a set $\{< PX_i, l_i >\}$, where PX_i is a PID-prefix with length l_i such that $l_0^1 < l_i \leq l_0^2$;

Initialize: let $\{< PX_i, l_i >\}$ be empty; let PX[k] be the *k-th* bit of PX; let $k_1 = l_0^1 + 1$; $k_e = l_0^2$;

Step 1: put $< PX_0^2, l_0^2 >$ into set $\{< PX_i, l_i >\}$;
Step 2: if $k_1 < k_e$, go to Step 3; otherwise, return set $\{< PX_i, l_i >\}$;
Step 3: $PX = PX_0^1$;
Step 4: $PX[k_1] = (PX_0^2[k_1] + 1) \bmod 2$;
Step 5: put $< PX, k_1 >$ into set $\{< PX_i, l_i >\}$;
Step 6: $k_1 = k_1 + 1$; go to Step 2;

combining G_1 with the skeleton, we obtain the inferred AS-level topology shown in Fig. 6(b), which is different from the actual AS-level topology shown in Fig. 6(a).

Prefix Conflict: The second type of error is prefix conflict. When adding AS paths into the inferred AS-level topology, we add a new node and an edge into the graph if two PID-prefixes are different. However, it is possible that one PID-prefix (e.g., 2.0.0.0/7) includes another one (e.g., 2.0.0.0/8). For instance, the AS-level topology shown in Fig. 7(a) has five two-hop AS paths. When considering the second hop *PIDs*, however, we can obtain only three PID-prefixes (i.e., namely 1.0.0.0/8, 4.0.0.0/8, and 9.0.0.0/8) and divide the *PID* sequences into three subsets. From the subset for the PID-prefix 1.0.0.0/8, we obtain the AS path (1.0.0.0/8, 2.0.0.0/7) because the two PID-prefixes 2.0.0.0/8 and 3.0.0.0/8 are aggregated into 2.0.0.0/7. Similarly, from the subset for the PID-prefix 4.0.0.0/8, we obtain the AS path (4.0.0.0/8, 2.0.0.0/7). But from the subset for PID-prefix 9.0.0.0/8, we obtain the AS path (9.0.0.0/8, 2.0.0.0/8). As a result, we obtain an inferred AS-level topology as shown in Fig. 7(b). Apparently, PX_1 2.0.0.0/8 is included by PX_0 2.0.0.0/7, which violates the rule that any two PID-prefixes assigned to inter-domain paths connecting an AS should be non-overlapping.

B. Correcting Errors Caused by Wrong Aggregation

Next we propose Algorithm 7 to correct the errors caused by wrong aggregation.

When requests are sent out by a_3 in Fig. 6, we have a correct AS path $\{$5.0.0.0/8, 3.0.0.0/8$\}$ from the *PID* sequences.

Algorithm 7: Correcting errors caused by wrong aggregation

Input: a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges; each edge is associated with a PID-prefix; the root node O of graph G; the set of *PID* sequences $\{\overline{PID}\}$ used for inferring G;
Output: a new graph $G' = (V', E')$, where each edge is associated with a PID-prefix and its corresponding prefix length;

Initialize: let $\mathcal{S} = \{\overline{PID}\}$; let S_v be a set of *PID* sequences stored by node v; initialize S_v to be empty for every node v in G; let Q be an empty queue;

Step 1: for every *PID* sequence in set $\mathcal{S}$, match it onto an AS path in graph G; if a *PID* sequence matches onto an AS path, put the *PID* sequence into a set corresponding to the node in the AS path that is furthest to node O;
Step 2: for every node v in G, compute the number of hops $h(v)$ from node O to v; put the leaf nodes into Q in the order of $h(v)$ such that the node with the largest $h(v)$ is at the head of Q and the one with the smallest $ith(v)$ at the tail of Q;
Step 3: if Q is empty, terminate; otherwise, let v be the head in Q, pop out v from Q, go to Step 4;
Step 4: if node v is a leaf node in G, go to Step 5; else, go to Step 3;
Step 5: let S_v be the set of *PID* sequences stored by node v; employ Algorithm 2 to infer an AS path $ASP_v = \{PX_i\}$ $(1 \leq i \leq h(v))$, with PX_1 being the first PID-prefix originated from node O;
Step 6: obtain the AS path from node O to node v in graph G, denote the AS path be $\overline{ASP}_v = \{\overline{PX}_i\}$, $1 \leq i \leq h(v)$;
Step 7: let ASP^t be a set of temporary AS paths and initialize it to be empty; let j be an index and initialize it to be 1;
Step 8: if PX_j is equal to $\overline{PX}_j$, go to Step 9; else, go to Step 10;
Step 9: put PX_j at the end of ASP^t, go to Step 13;
Step 10: take $\overline{PX}_j$ and PX_j as input, use Algorithm 6 to split $\overline{PX}_j$ into multiple PID-prefixes denoted by $\{PX_j^k\}$, k = 1, 2, $\cdots$, n;
Step 11: let $\{ASP_j\}$ be a set of AS paths and initialize it to be empty;
Step 12: for every AS path in ASP^t, put PX_j^k at the end of the AS path and form a new AS path, put the new AS path into the set $\{ASP_j\}$;
Step 13: empty the set ASP^t, let $ASP^t = \{ASP_j\}$;
Step 14: $j = j + 1$; if $j \leq h(v)$, go to Step 8; else, go to Step 15;
Step 15: if there is only one AS path in set ASP^t, go to Step 3; else, delete node v from the graph G, go to Step 16;
Step 16: use Algorithm 3 to add the AS paths in ASP^t into graph G;
Step 17: obtain the *PID* sequences $\mathcal{S}_1 = \{\overline{PID}\}$ maintained by nodes in AS path ASP_v;
Step 18: match all *PID* sequences in set $\mathcal{S}_1$ onto nodes in Graph G;
Step 19: check every leaf node in graph G, if the leaf node does not maintain any *PID* sequence, remove the leaf node and the corresponding edge from graph G;
Step 20: put the leaf nodes that are added into graph G into queue Q, go to Step 3.

Since the PID-prefix 3.0.0.0/8 is included by 2.0.0.0/7, we can use the AS path $\{$5.0.0.0/8, 3.0.0.0/8$\}$ to correct errors caused by a wrong aggregation. When expanding the skeleton, however, we ignore such important information. Therefore, we can exploit the original (instead of truncated) *PID* sequences to correct errors caused by wrong aggregation.

When PID-prefixes are aggregated by mistake, multiple AS paths merge to a single AS path. To address this issue, in Algorithm 6, we split a wrongly aggregated PID-prefix with a shorter prefix length (e.g., 2.0.0.0/7) into several PID-prefixes with longer prefix lengths (e.g., 2.0.0.0/8 and 3.0.0.0/8). In particular, given two PID-prefixes PX_0^1 and PX_0^2 with the lengths l_0^1 and l_0^2 ($l_0^1 < l_0^2$), Algorithm 6 uses the PID-prefix PX_0^2 as basis, and splits PX_0^1 into two 1-bit more specific PID-prefixes: one including PX_0^2 and the other not. For the one that includes PX_0^2, Algorithm 6 further splits it into two 1-bit more specific PID-prefixes until an obtained PID-prefix has the same length as that of PX_0^2. Note that "*mod*" in Step 4 means the modular operation. For instance, 1 *mod* 2 = 1, and 2 *mod* 2 = 0.

For example, given two PID-prefixes 192.168.0.0/13 and 192.168.0.0/16, Algorithm 6 first splits 192.168.0.0/13 into 192.168.0.0/14 and 192.172.0.0/14. Since 192.168.0.0/14 includes 192.168.0.0/16, Algorithm 6 further splits 192.168.0.0/14 into 192.168.0.0/15 and 192.174.0.0/15. Finally, Algorithm 6 splits 192.168.0.0/15 into 192.169.0.0/16 and 192.168.0.0/16, as illustrated by Fig. 8. As a result, Algorithm 6 splits the PID-prefix 192.168.0.0/13 into four PID-prefixes 192.172.0.0/14, 192.174.0.0/15, 192.168.0.0/16 and 192.169.0.0/16. One can see that the four PID-prefixes are non-overlapping and can be aggregated into 192.168.0.0/13. Note that while 192.172.0.0/14 and 192.174.0.0/15 can be further split into more specific PID-prefixes, we do not split them in Algorithm 6 because we want the split be conservative. However, later when it is necessary, they may be further split.

Algorithm 7 describes how to correct errors caused by wrong aggregations occurred in the inferred AS-level topology. The core idea behind Algorithm 7 is to find AS paths with more specific PID-prefixes (i.e., PID-prefixes with longer lengths) by using the *PID* sequences. To this end, Algorithm 7 first maps all *PID* sequences onto the nodes in the inferred AS-level topology. For example, a *PID* sequence {5.6.8.8, 3.1.5.8} can be mapped onto node b_3 in Fig. 6(b), and a *PID* sequence {4.8.8.8, 2.9.1.8} can be mapped onto node b_5 is Fig. 6(b). If a *PID* sequence is mapped onto a node, it is stored by the node. As a result, every node in the inferred AS-level topology should store many *PID* sequences.

Algorithm 7 then considers the *PID* sequences stored by every leaf node v in the initial AS-level topology and tries to obtain an AS path ASP_v from these *PID* sequences. We only consider leaf nodes because it is more likely to infer more specific PID-prefixes from *PID* sequences stored by leaf nodes. For example, we obtain the AS path ASP_{b3} = {5.0.0.0/8, 3.0.0.0/8} from the *PID* sequences stored by node b_3 in Fig. 6(b). Algorithm 7 then compares the obtained AS path ASP_v with the AS path $\overline{ASP}_v$ from the root node O to the leaf node in the inferred AS-level topology. If the two AS paths are the same, Algorithm 7 considers other leaf nodes; otherwise, it considers the first PID-prefix in ASP_v that is different from the PID-prefix in $\overline{ASP}_v$ with the same hop. For example, the AS path for node b_3 in Fig. 6(b) is $\overline{ASP}_{b3}$ = $\{5.0.0.0/8, 2.0.0.0/7\}$. Since ASP_{b3} is not equal to $\overline{ASP}_{b3}$, Algorithm 7 considers the first hop PID-prefixes 2.0.0.0/7 and 3.0.0.0/8. Taking the two PID-prefixes as input, Algorithm 7 splits the PID-prefix with the larger *PID* space into several more specific PID-prefixes by using Algorithm 6. For example, Algorithm 7 splits the PID-prefix 2.0.0.0/7 into two PID-prefixes 2.0.0.0/8 and 3.0.0.0/8.

Having obtained the more specific PID-prefixes, Algorithm 7 forms multiple sub-AS paths. For example, with PID-prefixes 2.0.0.0/8 and 3.0.0.0/8, we form two sub-AS path {2.0.0.0/8} and {3.0.0.0/8}. Since the second PID-prefix 5.0.0.0/8 of ASP_{b3} and that of $\overline{ASP}_{b3}$ are the same, Algorithm 7 adds it onto the sub-AS paths to form two complete AS paths {5.0.0.0/8, 2.0.0.0/8} and {5.0.0.0/8, 3.0.0.0/8}.

Algorithm 7 then removes the leaf node and corresponding edge from the inferred AS-level topology, and adds the obtained AS paths into the inferred AS-level topology. For example, after deleting node b_3 from Fig. 6(b) and adding the two AS paths {5.0.0.0/8, 2.0.0.0/8}, {5.0.0.0/8, 3.0.0.0/8}, we obtain the topology shown in Fig. 6(c).

Since some AS paths may not exist in the actual topology, Algorithm 7 removes such AS paths by checking whether there are *PID* sequences that could be mapped onto these AS paths. If no *PID* sequences are mapped onto an AS path, the AS path is removed from the topology. For example, since no *PID* sequences can be mapped onto the AS path {5.0.0.0/8, 2.0.0.0/8}, it is removed from the topology shown in Fig. 6(c). The resulting topology is shown in Fig. 6(d).

Algorithm 8: Correcting errors caused by prefix conflict

Input: a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges; each edge is associated with a PID-prefix; the root node O of graph G; the set of *PID* sequences $\{\overline{PID}\}$ used for inferring G;
Output: a new graph $G' = (V', E')$, where each edge is associated with a PID-prefix and its corresponding prefix length;

Initialize: $G' = G$; let $\mathcal{S} = \{\overline{PID}\}$; let S_v be a set of *PID* sequences stored by node v; initialize S_v to be empty for every node v in G'; let Q be an empty queue; let $h(v)$ be the number of hops from node O to node v in the graph G';

Step 1: for every *PID* sequence in set $\mathcal{S}$, match it onto an AS path in graph G'; if a *PID* sequence matches onto an AS path, put the *PID* sequence into a set corresponding to the node v in the AS path such that $h(v)$ is the largest among all nodes in the AS path;
Step 2: for every non-leaf node v in G', put the triple $< v, h(v), S_v >$ into queue Q such that, for two nodes v_1 and v_2, v_1 is in front of v_2 in Q if $h(v_1) > h(v_2)$;
Step 3: if Q is empty, terminate; otherwise, take out the first element $< v, h(v), S_v >$ from Q;
Step 4: put every neighbor node u of v into a set N_v if $h(u) > h(v)$; for every node u in N_v, put the triple $< PX_{vu}, v, u >$ into a set $\overline{PX}_v$, where PX_{vu} is the PID-prefix for the edge (v, u);
Step 5: if the set $\overline{PX}_v$ is empty, go to Step 3; otherwise, go to Step 6;
Step 6: check whether or not there is a prefix conflict in set $\overline{PX}_v$. If no prefix conflict exists, go to Step 3; otherwise, denote the two *PID* -prefixes by PX_1 and PX_2, with PX_1 has shorter PID-prefix length than PX_2; denote the edge for PX_1 by (v, u) and let G_u be the subgraph rooted by node u in G'; remove G_u and edge (v, u) from G', go to Step 7;
Step 7: take PX_1 and PX_2 as input, use Algorithm 6 to split PX_1 into a set of PID-prefixes denoted by $\{PX_2, PX_1^1, PX_1^2, \cdots, PX_1^m\}$;
Step 8: for each PID-prefix in set $\{PX_1^1, PX_1^2, \cdots, PX_1^m\}$, add a node u_i into graph G', add an edge (v, u_i) into G', and add G_u into G', with the root of G_u being node u_i; add node u_i into Q;
Step 9: denote the edge for PX_2 by (v, w), add G_u into G' with the root node being w;
Step 10: put the *PID* sequences stored by nodes in subgraph G_u in a set $\{\overline{PID}_u\}$; match *PID* sequences in $\{\overline{PID}_u\}$ onto nodes in graph G';
Step 11: check every leaf node in graph G', if the leaf node does not maintain any *PID* sequence, remove the node and the corresponding edge from graph G';
Step 12: add node v into Q, go to Step 3.

C. Correcting Errors Caused by Prefix Conflict

Next we propose Algorithm 8 to correct errors caused by prefix conflict.

For every non-leaf node v in the inferred AS-level topology G, we check whether there is a prefix conflict among the PID-prefixes for edges connecting node v to its neighbors in G. If there is a prefix conflict between two PID-prefixes PX_1 and PX_2, without losing generality, we assume that PX_1 has the shorter prefix length. In addition, the edge of PX_1 connects the node v to the node u in G, and the edge of PX_2 connects the node v to the node w in G. We then record the subgraph G_u in G rooted with the node u, delete the subgraph G_u and the edge

TABLE I
TOPOLOGIES USED IN OUR SIMULATIONS.

	top 1	top 2	top 3	top 4
number of *ASes*	3,000	5,000	10,000	20,000
number of inter-domain paths	5,444	9,169	18,294	35,942

(v, u) from G. After that, we use Algorithm 6 to split PX_1 into multiple PID-prefixes denoted by $\{PX_2, PX_1^1, PX_1^2, \cdots, PX_1^m\}$. In addition, for each PID-prefix PX_1^i, we add a new node u_i and a new edge (v, u_i) for PX_1^i into the graph G, and add the subgraph G_u into G with the root of G_u being the node u_i. Furthermore, we also add the subgraph G_u into G with the root of G_u being the node w.

Consider the example shown in Fig. 7(b). When checking the node v, Algorithm 8 first finds that there is a prefix conflict. Accordingly, it splits the PID-prefix 2.0.0.0/7 into 2.0.0.0/8 and 3.0.0.0/8. After deleting the subgraph G_u and the edge (v, u) from the topology, Algorithm 8 adds the node u_1 and the edge (v, u_1) into the graph. In addition, it adds the subgraph G_u into the graph by letting u_1 being the root of G_u, as shown at the bottom of Fig. 7(c). Furthermore, Algorithm 8 adds the same subgraph G_u into the graph by letting w as the root of G_u, as shown on the right of Fig. 7(c).

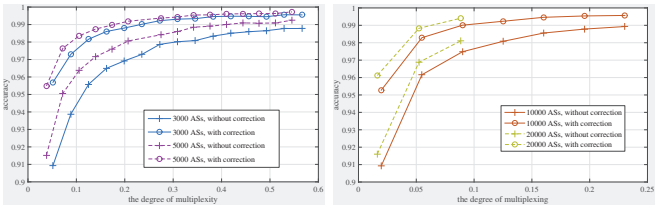
As in Algorithm 7, some AS paths in G may not exist in the actual topology. Algorithm 8 thus removes such AS paths by checking whether there are *PID* sequences that could be mapped onto these AS paths. If no *PID* sequences are mapped onto an AS path, the AS path is removed from the topology.

To obtain the final AS-level topology, we need to run Algorithm 7 and Algorithm 8 interchangeably until there is no change in the topology. For instance, after using Algorithm 7 to correct errors caused by wrong aggregations, there is an error caused by prefix conflict because PID-prefixes 3.0.0.0/8 and 2.0.0.0/7 are overlapped, as shown in Fig. 6(d). Accordingly, we need to use Algorithm 8 to correct such an error.

VII. SIMULATION RESULTS

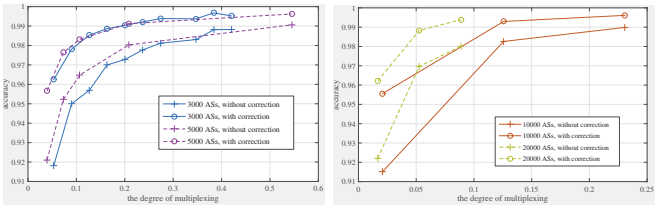
To evaluate the accuracy of the proposed algorithms for inferring the AS-level topology, we conduct extensive simulations over four Internet-like topologies, which are generated with the method in [26] and shown in Table I. Although the current Internet has more than 60,000 *ASes*, we simulate up to 20,000 *ASes* because of two considerations. First, a large network topology will cause very high simulation time. Second, as shown in this section, the inferring accuracy of our scheme increases with the topology size. For every topology, there is an observer located in an AS and nodes in other *ASes* send requests to the observer so that the observer can collect many *PID* sequences. For each case, we conduct ten simulations. In each simulation, the PID-prefixes are randomly assigned to inter-domain paths. The results are the average of ten simulations.

We use two performance metrics to evaluate our proposed algorithms under different conditions. The first metric is the accuracy of inferring AS-level topology. For a given AS-level network topology with m inter-domain paths, every inter-domain path is assigned a PID-prefix, and there are m PID-prefixes in the network. We infer the AS-level topology and the corresponding PID-prefix for each inter-domain path. If the inferred inter-domain path and the corresponding PID-prefix is the same as the one in original AS-level topology, the inferred PID-prefix is considered to be correct. The inferring accuracy is defined to be the ratio of the number of correct PID-prefixes inferred by our algorithms over that of the total number (i.e., m) of PID-prefixes in the actual AS-level topology.



(a) Network size = 3,000 or 5,000 (b) Network size = 10,000 or 20,000

Fig. 9. The inferring accuracy of the proposed approach when the length of PID-prefixes is 16 and every AS sends out 500 requests.



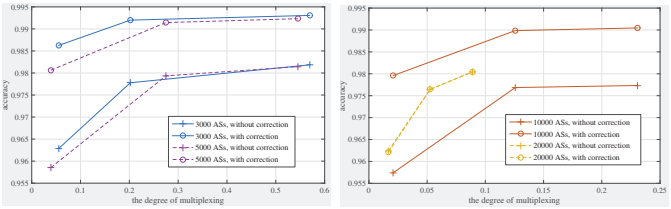
(a) Network size = 3,000 or 5,000 (b) Network size = 10,000 or 20,000

Fig. 10. The inferring accuracy of the proposed approach when the length of PID-prefixes is 16 and an AS sends out 500 - 5,000 requests.

Since the inferring accuracy is not 100%, the second performance metric that we are interested in is where the errors occur. Generally, an observer prefers an error occurred at the edge of the inferred AS-level topology over the one occurred at the core of the inferred AS-level topology, as the later can impact more paths. For example, the observer can traceback an attacker based on the inferred AS-level topology and a set of new *PID* sequences. In this case, it is preferred that the errors occur at the first hop such that the observer can traceback the attacker closely. Fortunately, for all our simulations, the errors occur at the first hop of the inferred and the actual AS-level topology if Algorithms 6 - 8 are employed. Such errors may occur when two AS paths share the rest but the first hop (e.g., in Fig. 5(d), AS paths O-v_{61}-v_{63} and O-v_{61}-v_{62}), while the first hop PID-prefixes can be aggregated.

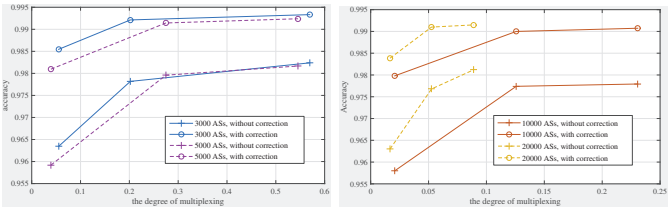
We study the effects of the following factors on the accuracy of our proposed approach:

- *Distribution of PID-prefix lengths*: We consider two types of length distribution. In the first case, all PID-prefixes have the same length (i.e., 16 bits). In the second case, the lengths of PID-prefixes are either 11 bits or 16 bits. In both cases, the observer does not know in priori the length of PID-prefixes.
- *Traffic distribution*: the number of *PID* sequences generated by each AS in the simulated AS-level topology. We also consider two cases. In the first case, every AS



(a) Network size = 3,000 or 5,000 (b) Network size = 10,000 or 20,000

Fig. 11. The inferring accuracy of the proposed approach when the length of PID-prefixes is either 11 or 16 and an AS sends out 500 requests.



(a) Network size = 3,000 or 5,000 (b) Network size = 10,000 or 20,000

Fig. 12. The inferring accuracy of the proposed approach when the length of PID-prefixes is either 11 or 16 and an AS sends out 500-5,000 requests.

generates 500 requests (thus 500 *PID* sequences). In the second case, an AS randomly generates 500 - 5,000 requests towards the observer.

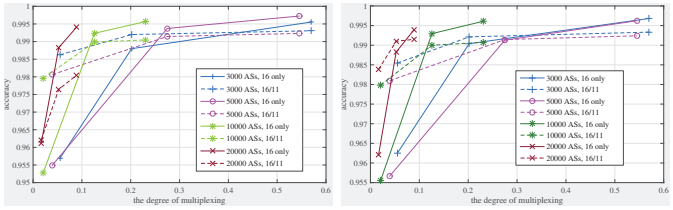
- *Degree of multiplexing*: the ratio of the number of unique PID-prefixes assigned to inter-domain paths over the number of inter-domain paths in the AS-level topology. Apparently, if we assign a unique PID-prefix to every inter-domain path, the degree of multiplexing is 1. On average, a lower degree of multiplexing means that a PID-prefix is assigned to (thus shared by) more inter-domain paths. In order to avoid the prefix conflict in the AS-level topology, the degree of multiplexing cannot be too small.

A. Performance Improvement with Error Correction

Figures 9 - 12 show the inferring accuracy of the inferred AS-level topology under different parameter settings when Algorithms 6 - 8 are employed (marked as "*with correction*") and not employed (marked as "*without correction*"). Almost in all cases, Algorithms 6 - 8 can correct some errors and obtain higher accuracy. For example, from Fig. 9(a) we can observe that Algorithms 6-8 can improve the inferring accuracy from 90.93% to 95.69% when the degree of multiplexing is about 0.052 and the network size is 3,000. Similarly, when the network size is 5,000 and the degree of multiplexing is 0.038, the inferring accuracy is improved from 91.52% to 95.49% if Algorithms 6-8 are employed.

In all cases, the improvement in the inferring accuracy reduces when the degree of multiplexing increases. For instance, the inferring accuracy is improved from 98.78% to 99.55% when the degree of multiplexing is about 0.566 and the network size is 3,000, which implies an improvement of 0.77%. By contrast, the improvement is 4.76% when the degree of multiplexing is about 0.052. This is because a higher degree of multiplexing implies that more PID-prefixes are assigned to inter-domain paths when other parameters are fixed. This in turn leads to smaller probability of wrong aggregation and less prefix conflict.

More importantly, for all simulated cases, the inferring accuracy is higher than 90% even if we do not use Algorithms 6-8 to correct some errors, and the inferring accuracy is higher than 95% if Algorithms 6-8 are employed.



(a) Each AS sends 500 requests (b) Each AS sends 500-5000 requests

Fig. 13. The effect of the distribution of PID-prefixes on the inferring accuracy of the proposed algorithms.

B. Effect of the Degree of Multiplexing

From Fig. 9 to Fig. 12, we observe that the degree of multiplexing affects the accuracy of our algorithms. Generally, a larger degree of multiplexing leads to higher accuracy. For instance, in Fig. 9, if the network size is 5,000 and Algorithms 6-8 are employed, the accuracy is 95.49% when the degree of multiplexing is 0.038. By contrast, when the degree of multiplexing is 0.545, the accuracy increases to 99.72%. This is also because a higher degree of multiplexing implies that more PID-prefixes are assigned to inter-domain paths, which leads to a smaller probability of prefix conflict.

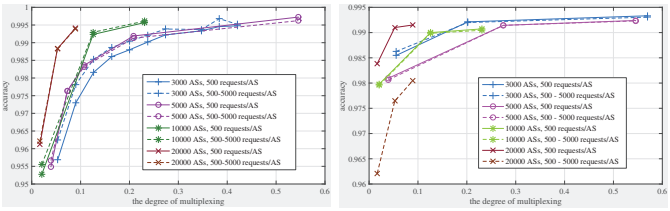
C. Effect of the Distribution of PID-Prefixes

In Fig. 13, the accuracy is higher than 95%, regardless whether or not the length of PID-prefixes changes. However, the distribution of PID-prefixes does affect the accuracy of our proposed approach. In particular, when the degree of multiplexing is low, the accuracy when all PID-prefixes have the same prefix length of 16 bits is lower than that when the lengths of PID-prefixes are either 16 or 11 bits, because it is easier to tell apart two PID-prefixes with different lengths than two with the same length.

On the other hand, when the degree of multiplexing is high, the accuracy when all PID-prefixes have the same prefix length of 16 bits is higher than that when the lengths of PID-prefixes are either 16 or 11 bits. This is because, in this case, the number of unique PID-prefixes assigned to inter-domain paths becomes larger. However, since the length of *PIDs* is fixed to be 32 bits, it is more likely that two PID-prefixes are aggregable when the number of employed unique PID-prefixes becomes larger.

D. Effect of the Traffic Distribution

Fig. 14 shows that the accuracy is higher than 95% for all simulated network topologies, regardless of the length of PID-prefixes and the traffic distribution. While we observe from



(a) Length of PID-prefix = 16 (b) Length of PID-prefix = 11 or 16

Fig. 14. The effect of the traffic distribution on the inferring accuracy of the proposed algorithms.

Fig. 14 that the traffic distribution does affect the accuracy, the difference of different distribution is very small since in most cases, the two curves with the same marker is quite close.

VIII. Conclusion

In this paper, we propose a set of strategies to infer the AS-level topology of an Internet deploying CoLoR, which uses cryptographic *PIDs* for inter-domain routing and packet forwarding. We conduct extensive simulations over four network topologies by considering different parameters including the degree of multiplexing, the distribution of PID-prefixes, and the traffic distribution. The results show that the proposed approach can achieve an inferring accuracy higher than 95% and all errors occur at the edge of the inferred AS-level topologies.

References

[1] M. Handley, "Why the Internet only just works," *BT Tech. J.*, vol. 24, no. 3, pp. 119 - 130, Jul. 2006.

[2] A. Feldmann, "Internet clean-slate design: what and why?", *ACM CCR*, vol. 37, no. 3, pp. 59 - 64, Jul. 2007.

[3] J. McCauley, Y. Harchol, A. Panda, *et al.*, "Enabling a permanet revolution in Internet architecture," in *Proc. SIGCOMM'19*, Aug. 2019, Beijing, China.

[4] D. Trossen, M. Sarela, K. Sollins, "Arguments for an information-centric internetworking architecture," *ACM CCR*, vol. 40, no. 2, Apr. 2010, pp. 27 - 34.

[5] D. G. Andersen, H. Balakrishnan, N. Feamster, *et al.*, "Accountable internet protocol (AIP)," in *Proc. SIGCOMM'08*, Aug. 2008, Seattle, WA, USA.

[6] D. Barrera, L. Chuat, A. Perrig, *et al.*, "The SCION Internet architecutre," *Commun. ACM*, vol. 60, nol. 6, Jun. 2017, pp. 56 - 65.

[7] T. Koponen, S. Shenker, H. Balakrishnan, *et al.*, "Architecting for innovation," *ACM CCR*, vol. 41, no. 3, pp. 24 - 36, Jul. 2011.

[8] D. Han, A. Anand, F. Dogar, *et al.*, "XIA: efficient support for evolvable internetworking," in *Proc. NSDI'12*, Apr. 2012, San Jose, CA, USA.

[9] T. Koponen, M. Chawla, B. C G. Chun, et al, "A data-oriented (and beyond) network architecture," in *Proc. SIGCOMM'07*, Aug. 2007, Kyoto, Japan.

[10] L. Zhang, A. Afanasyev, J. Burke, et al, "Named data networking," *ACM CCR*, vol. 44, no. 3, pp. 66 - 73, Jul. 2014.

[11] G. Xylomenos, C. N. Ververidis, V. A. S. N. Fotiou, *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surv. & Tut.*, vol. 16, no. 2, Jul. 2013, pp. 1024C1049.

[12] H. Luo, Z. Chen, J. Cui, *et al.*, "CoLoR: an information-centric internet architecture for innovations," *IEEE Network*, vol. 28, no. 3, pp. 4 - 10, May 2014.

[13] H. Luo, Z. Chen, J. Li, *et al.*, "Preventing distributed denial-of -service flooding attacks with dynamic path identifiers," *IEEE Trans. on Inf. Foren. and Sec.*, vol. 12, no. 8, pp. 1801 - 1815, Aug. 2017.

[14] H. Luo, Z. Chen, J. Li, *et al.*, "On the benefits of keeping path identifiers secret in future Internet: a DDoS perspective," *IEEE Trans. on Netw. and Serv. Man.*, vol. 15, no. 2, pp. 650 - 664, Jun. 2018.

[15] Z. Chen, H. Luo, M. Zhang, *et al.*, "Improving Network Security by Dynamically Changing Path Identifiers in Future Internet," in *Proc. IEEE Globecom'15*, Dec. 2015, San Diego, CA, USA.

[16] BGP Reports. Available at: https://bgp.potaroo.net/index-bgp.html.

[17] AS Rank: A ranking of the largest Autonomous Systems (AS) in the Internet. Available at: https://asrank.caida.org/asns.

[18] H. Luo, Z. Chen, J. Cui, *et al.*, "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.

[19] H. Luo, J. Cui, Z. Chen, *et al.*, "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.

[20] M. Zhang, H. Luo, H. Zhang, "A Survey of Caching Mechanisms in Information-Centric Networking," *IEEE Commun. Surv. & Tut.*, vol. 17, no. 3, Aug. 2015, pp. 1473 - 1499.

[21] J. Li, H. Luo, S. Zhang, *et al.*, "Traffic engineering in information-centric networking: opportunities, solutions and challenges," *IEEE Commun. Mag.*, vol. 56, no. 11, pp. 124 - 130, Nov. 2018.

[22] J. Li, H. Luo, S. Zhang, *et al.*, "Design and implementation of efficient control for incoming inter-domain traffic with information-centric networking," *J. of Netw. and Comput. Appl.*, vol. 133, pp. 109 - 125, May 2019.

[23] R. Motamedi, R. Rejaie, and W. Willinger, "A survey of techniques for Internet topology discovery," *IEEE Commun. Surv. & Tut.*, vol. 17, no. 2, pp. 1044 - 1065, Apr. 2015.

[24] Y. Jin, C. Scott, A. Dhamdhere, *et al.*, "Stable and Practical AS Relationship Inference with ProbLink," in *Proc. NSDI'19*, Feb. 2019, Boston, MA, USA.

[25] W. G. Cochran, *Sampling Techniques*, 3rd ed. John Wiley & Sons, 1977.

[26] J. Tomasik, M. Weisser, "aSHIIP: autonomous generator of random Internet-like topologies with inter-domain Hierarchy," in *Proc. IEEE MASCOTS'10*, Aug. 2010, Miami Beach, FL, USA.