IPD: Detecting Traffic Ingress Points at ISPs

Stefan Mehner
University of Kassel
Germany

Helge Reelfs
Brandenburg University of Technology
Germany

Ingmar Poese
BENOCS
Germany

Oliver Hohlfeld
University of Kassel
Germany

ABSTRACT

Detecting *where* traffic enters a network enhances network operation, but poses a complex measurement problem that requires analyzing a continuous traffic stream from *all* border routers—a challenging task for ISPs in the absence of a scalable approach.

To enable ISPs to perform Ingress Point Detection (*IPD*), we propose an efficient approach that accurately identifies traffic ingress points at ISPs of any size using flow-level traffic traces. *IPD* identifies the specific router and interface through which a particular segment of the Internet address space enters a network. *IPD* splits the address space into fine-grained ranges to identify the specific router and interface through which segments of the address space enter the network. We have deployed *IPD* for six years at a major tier-1 ISP with an international network that handles multi-digit Tbit/s traffic levels, and our experience shows that *IPD* can accurately identify ingress points and scale to high traffic loads on a single commodity server. *IPD* enabled the ISP to improve network operations by identifying performance issues and realizing advanced traffic engineering practices.

CCS CONCEPTS

• **Networks** → **Network manageability**; **Network measurement**; **Network dynamics**.

KEYWORDS

traffic engineering, inter-domain traffic, operational experience

ACM Reference Format:
Stefan Mehner, Helge Reelfs, Ingmar Poese, and Oliver Hohlfeld. 2024. *IPD*: Detecting Traffic Ingress Points at ISPs. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3651890.3672232

1 INTRODUCTION

"Why is Netflix slow at home in only one city of an ISP's network?" Answering such seemingly simple questions poses a complex traffic analysis problem. Even if the issue is not the ISP's fault, they are often blamed since they provide upstream to their customers. Therefore, it is valuable for ISPs to identify and resolve performance bottlenecks and other traffic-related issues to maintain customer satisfaction. This requires understanding where traffic enters the network, which is a complex problem, as we will show.

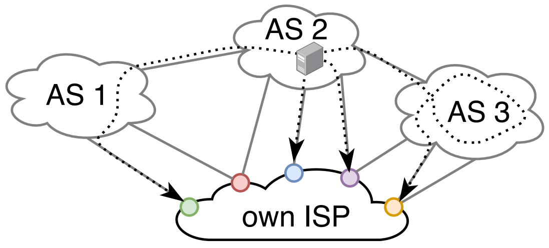


Figure 1: Paper Goal: Enable ISPs to infer *where* traffic enters their network (colored ingress points). Detecting them requires analyzing continuous traffic streams from *all* border routers—a complex analytics problem.

ISPs face numerous operational questions that necessitate performing traffic ingress point detection. For instance, if an ISP's expensive intercontinental links are becoming fully loaded, they need to determine the origin of the traffic being carried over these links to decide if investing in additional capacity would be profitable. ISPs can check if their peers are violating settlement-free peering agreements by carrying traffic from other peers over peering links.

Answering such operational questions in small networks is easy, but complicated in larger ones for two reasons. First, large networks have multiple interconnection links with other ASes, leading to several possibilities for traffic to enter the ISP's network (see Figure 1). These possibilities depend on routing policies and routing decisions made by *other* networks. Second, while BGP determines the forward path (i.e., from the ISP to the destination AS), it cannot determine the reverse path due to asymmetries [3, 11, 12]. BGP can only eliminate routes by selective announcements or make them less likely with egress traffic engineering. Still, it cannot definitively determine where traffic enters an ISP's network—even though practitioners sometimes attempt to do this.

This paper introduces a data-driven solution that enables ISPs to determine traffic ingress points for any prefix. Our *IPD* algorithm is scalable and can be used by ISPs of any size to identify traffic ingress points online. *IPD* solves a complex measurement problem by analyzing flow-level traffic traces from *all* border routers of an ISPs network. It runs at fine-granular time-bins, is configurable to as little as minutes, and can run on readily available commodity hardware, requiring only a single server for an entire ISP. We have deployed *IPD* at a major tier-1 ISP and evaluated its performance

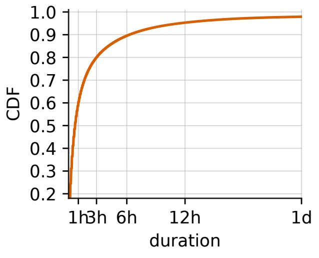


Figure 2: Stability duration per prefix on a link: 60% remain stable for < 1 hour.

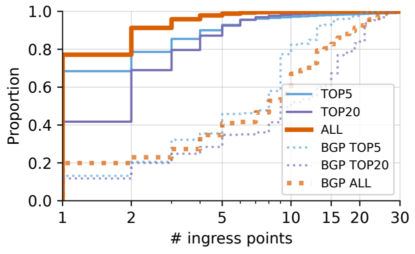


Figure 3: Ingress router count per prefix: more BGP paths exist, while traffic of most prefixes only have 1 ingress point.

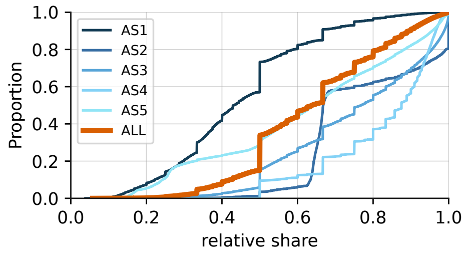


Figure 4: Relative traffic share of first ranked ingress router per /24 prefix: often a single ingress point is dominant.

over the course of six years. We share our deployment experience and contribute the following:

- We present *IPD*, an online algorithm to detect traffic ingress points at ISPs of any size. It solves a complex traffic analysis problem by analyzing incoming traffic flows at *all* border routers of an ISPs' network—at any traffic rate and in real-time (§ 3).
- We show that the studied tier-1 ISP exhibits high traffic dynamics and that ingress points can thus often change. *IPD* thus needs to run in minute intervals at large networks on commodity hardware (one server for the studied tier-1 ISP). To run the approach at scale, we present design choices enabling scalability and further determine the *IPD* parameterization with a systematic parameter study that assessed over 300 different parameter sets.
- We have deployed *IPD* at a major tier-1 ISP for six years and use this experience to empirically evaluate the proposed *IPD* algorithm (§ 5). The ISP operates a large international network and shifts traffic in the multi-digit TBit/s range, which shows the scalability of the approach. We show that it accurately infers traffic ingress points, how traffic dynamics impact the stability of ingress mappings, the asymmetry of Internet paths by correlating ingress with egress points (BGP), and operational use cases.
- We release a prototypical *IPD* implementation at [24]. To quickly experiment with *IPD*, we further release a Mini IPD environment [25] that runs *IPD* in the Mini Internet [14] and includes an example ISP scenario—ready to be used for research and teaching.

2 INGRESS POINTS CHANGE OFTEN

Internet traffic is highly dynamic and traffic ingress points change quickly, as we will show in this section. These changes pose a challenge to network operation, in particular traffic engineering. Traffic dynamics and sudden changes in ingress points originate from changes in CDN mapping functions (server selection), demand variability, server maintenance, or BGP/IGP route adjustments. For improved network operation, ISPs can benefit from ingress point information—a challenging problem that is addressed by this work.

Ingress points change quickly at ISPs. We begin by motivating that it is insufficient to run *IPD once* or to use static rules. Due to the traffic dynamics—primarily originating from CDN operations—ISPs must *regularly* run *IPD* (e.g., every n minutes). To address this challenge, our *IPD* algorithm (§ 3), efficiently processes traffic from all border routers within minutes. We remark that this paper focuses on *IPD* at ISPs that largely face ingress traffic dynamics imposed by server selection at cloud and CDN providers (ingress dynamics at CDNs/clouds can differ!).

To study traffic dynamics at a tier-1 ISP (see § 4), we quantify the duration of prefix stability at ingress points using the raw *IPD* results (Figure 2). Notably, 60% of prefixes remain stable for less than one hour, while only 10% remain stable for more than six hours. This highlights that ingress points change quickly. To use *IPD* for improved network operation (e.g., CDN traffic steering [28]), the *IPD* needs to run frequently.

Most prefixes only have one ingress point. Next, we investigate the number of potential ingress points into the tier-1 ISP's network. Figure 3 shows the distribution of the number of different next-hop routers for each prefix from BGP tables (dotted lines). The solid lines illustrate the number of simultaneous ingress points per /24 prefix, derived from the ISP's flow traffic data (§ 4). We further show the prefixes of the top 5 and top 20 ASes that contribute the most ingress traffic to the ISP. For BGP, we observe that only 20% of the prefixes have only one next-hop router, while 60% have more than five possible routes and thus likely ingress points. We observe similar trends for the top 5 and top 20 ASes by overall traffic.

Yet, when examining the actual traffic ingress points extracted from the flow data, it becomes evident that traffic does not enter via all the BGP-announced links. Instead, nearly 80% of the traffic enters through only one ingress point when considering the entire distribution. For the top 5 and top 20 ASes, about 30% and 58% of the cases, respectively, involve multiple entry points. As a result, in most cases, only a single ingress point exists.

Single ingress points carry the bulk of traffic. In Figure 4, we focus on prefixes that have more than one ingress point. This CDF shows the distribution of traffic share across different ingress links. The orange curve represents the traffic distribution across all ASes, and the various shades of blue represent the top 5 ASes' distributions. For 80% of all prefixes, 80% or less of the traffic enters through the primary ingress link. In other words, a dominant ingress point exists that carries the bulk of the traffic.

However, when multiple ingress points have a significant traffic share, traffic engineering becomes more complex. In this case, it's not sufficient to detect the ingress point (source IP to ingress link mapping). Destination networks must also be tracked to determine the path of flows through the ISP network and identify the issue. This quickly results in a state explosion, which *IPD* avoids by not tracking complete paths.

Takeaway. *Our empirical motivation shows the fundamental design requirements for our IPD algorithm. Firstly, ingress points change frequently and thus the IPD algorithm needs to run frequently, e.g., every minute. Processing large data volumes from all border routers within minutes poses a big data problem. Secondly, for most prefixes only a single ingress point exists. In case of multiple ingress points, typically a single carries the bulk of the traffic per prefix (e.g., 80%). Since these are the ingress points that traffic engineering is performed on, our algorithm needs to additionally identify the ingress point that carries the most traffic.*

3 INGRESS POINT DETECTION (*IPD*)

We next describe the design of *IPD*, which is designed to detect the ingress points at the smallest configured prefix granularity ($cidr_{max}$). It thus executes a traffic-based partitioning of the IP address space unrelated to how BGP partitions the space.

3.1 Design Aspects and Requirements

Identifying network ingress traffic requires processing data across all border routers. Our observations highlight key design considerations for our *IPD* algorithm:

Ingress prefixes. The *IPD* focuses on inferring and clustering ingressing prefixes (from the source addresses) by traffic (not BGP!).

BGP is not an option. Despite popular belief in the operations community, BGP data cannot be used to (more easily) detect ingress points. While BGP tables offer a comprehensive view of AS interconnectivity and manageable data sizes, they only dictate traffic's outbound paths, while the sending party has the power to change inbound routes, often leading to path asymmetry. Further, BGP prefixes are not aligned with *IPD* prefixes, which are typically more specific. These limitations render BGP unsuitable for ingress point detection, as we illustrate in § 5.5.

Input data: sampled flow-level traffic. *IPD* is a traffic-based clustering of the IP address space; thus, the input data is traffic. For mapping source IP addresses to ingress links, we rely on flow-level traces (e.g., Netflow or IPFIX) from all border routers. It is important to note that only traffic from ingress links can be used as input. This *IPD* process requires handling substantial data volumes (in our case, hundreds of border routers with the total traffic exceeding tens of Tbit/s). To cope with these high data rates, routers apply random packet sampling (1 out of n pkts) with rates that range from $n = 1,000$ to $10,000$ (depending on vendor, software, etc.). Thus, unsampled data is *never* available.

Addressing clock drift with statistical time. While router clocks would always be perfectly synchronized in an ideal setting, the deployment practice on $> 3,000$ routers shows that inaccurate router clocks occur. Inherent discrepancies in router clocks thus necessitate a robust method to address potential data errors like inaccurate timestamps. We address this issue in a pre-processing step. Since this step is not part of the *IPD*, we only briefly sketch the approach and remark that its details are beyond the scope of this work. To do so, we rely on inferring sequences of events from time input in the flow data, rather than assuming that all clocks are in sync. This *statistical time* approach segments traffic into uniform time buckets and analyzes flow samples within these periods. Intervals that don't meet a certain activity threshold are discarded, along with data outside the current time range. This method might exclude some data but ensures consistency despite clock drifts.

Efficiency and scalability. With hundreds of border routers generating millions of flow records per minute, *IPD* needs to scale. We made the following design choices as *IPD* is an online algorithm that must be completed by the end of each time bucket (e.g., 1 minute).

1) Aggregating IPs into *IPD* ranges. For effective ingress point detection, we employ a *traffic-based* aggregation using the CIDR addressing scheme, avoiding the inefficiency of tracking every individual IP address. This method treats the Internet's address space as a binary tree, with each node representing a CIDR range. To streamline the process, we mask each source IP with a predefined maximum CIDR mask ($cidr_{max}$), significantly reducing the number of individual addresses to monitor. Since BGP prefixes are not usable (see § 5.2), we implement a *traffic-based* aggregation based on the CIDR addressing scheme. This approach allows for manageable state maintenance while ensuring accurate *IPD*.

2) Optional simplification: Preferring flow counts over byte counts. A direct implementation of *IPD* will likely be based on using byte counts. For efficiency reasons, we use byte counts for classification of *IPD* ranges and switch to counting flows once an ingress link for a *IPD* range is classified (i.e., $s_{ipcount}$ is flow-based). Our primary motivation is to lower the frequency of counter overflows. A 32-bit byte counter would quickly overflow on high-capacity links with normal loads. We experimented with representing large integers (bigint) to prevent this, but it significantly slowed down all computations involving the per-range sample counts. However, we did not encounter the same issue of counter overflows that were too quick when using flow counts in the per-range fields. In our case, we observe a strong correlation (0.82) between flow and byte counts in our traffic, which shows that flow counts can serve as a proxy measure. Our practical deployment of IPD classification using flow counts instead of byte counts yielded good results. Users of *IPD* with other requirements might opt not to use this simplification and base all counters on byte counts directly.

Real-time detection. As we show in § 2, 60% of the prefixes remain stable at the same ingress link for less than one hour. Since accurate traffic engineering decisions can only be made based on actual data, the algorithm needs to operate online. Thus, a parameter t that defines the length of a time bucket can adjust the frequency of ingress point updates.

Focus on dominant ingress points. Our approach necessitates identifying *all possible* ingress points for each prefix carrying enough traffic, along with their traffic shares. This data is valuable for ISPs, as it helps them identify incorrect CDN mappings and enhances traffic engineering via ISP-CDN collaboration [28]. Many of these network management scenarios, including traffic steering and debugging, focus on the dominant ingress point carrying the bulk of the traffic (per prefix). Yet, most prefixes usually have one dominant ingress point or a primary link handling most traffic (§ 2), as evidenced by our six-year deployment experience at a tier-1 ISP. Thus, our algorithm operates in two stages to manage this. Initially, it identifies all potential ingress points per prefix. Subsequently, it narrows down to those that are unique or handle most of the traffic for each *IPD* range. To make the algorithm robust to handle

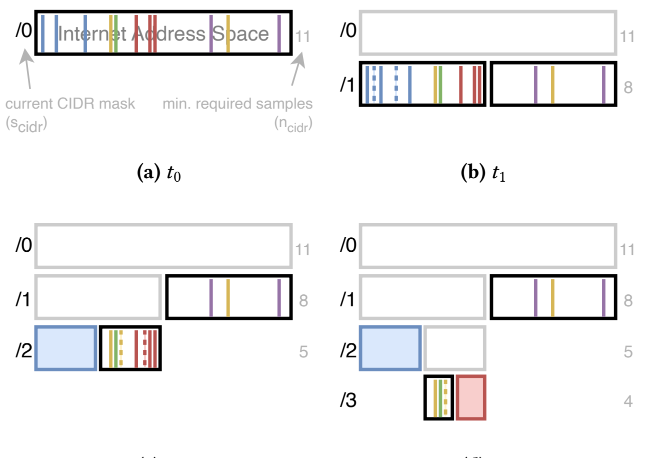

/0
Internet Address Space
11
current CIDR mask (s_{cidr})
min. required samples (n_{cidr})
/1
8
/2
5
/3
4
(a) t_0 **(b)** t_1 **(c)** t_2 **(d)** t_3

Figure 5: *IPD* algorithm: example application

a proportion of abnormal or incorrect (e.g., spoofed[1]) traffic, a configurable quality parameter q exists. In our operational setting, the default value for $q = 0.95$ means that 5% of the traffic for that prefix may ingress over different links.

Focus on high-traffic prefixes. The *IPD* is designed to detect the ingress points of high-traffic prefixes at the smallest granularity to steer or debug traffic that is of (high) operational relevance to the ISP. Omitting to detect ingress points for prefixes that hardly carry any traffic is thus an accepted consequence of our design.

3.2 *IPD* Algorithm

Our *IPD* algorithm utilizes a *traffic-based* partitioning approach that divides the IP address space into dynamic *IPD* ranges at high granularity, adapting to the dynamics of ingress traffic. It employs a top-down strategy for efficiency, starting with the entire IP address space, represented as /0 CIDR range. This process, akin to a 'divide and conquer' method, involves sequentially subdividing the IP range into smaller segments. Each subdivision or classification occurs only when sufficient traffic is detected, and the minimum sample count (n_{cidr}) is met for a given segment. This count is directly related to the segment's CIDR mask size, ensuring that classification decisions are based on sufficient data (larger prefixes need more samples than smaller ones).

Example illustrating the algorithm. To show the operation of the *IPD* algorithm, we show the IPv4 address space as a rectangle ranging from 0.0.0.0 to 255.255.255.255 in Figure 5a. Initially, the algorithm does not know existing network segments. As network traffic enters, source IP addresses from flow data are added to the entire /0 range, represented as lines within the rectangle (Figure 5a). The line color corresponds to the different ingress points. In our minimal example, n_{cidr} values are on the right side of the range ($n_{cidr} = 11$ for /0). Since 11 flows (called $s_{ipcount}$) are entered at t_0, the minimum number of required samples is reached (line 8 in the listing), and the algorithm can check if one dominant ingress point (color) exists ($s_{ingress} \geq q$). As many ingress points (colors) exist at the moment, the algorithm splits the range into two equal /1 ranges (0.0.0.0/1 and 128.0.0.0/1) at t_1, as can be seen in Figure 5b. By the time t_1, two new samples have been added to the left /1 range indicated by the dashed line style. At this point, the current sample count (s_{cidr}) is 10 and exceeds the minimum required samples for that range ($n_{cidr} = 8$). Yet, no prevalent ingress can be detected since 4 colors exist (line 9), which results in a further split as can be seen in Figure 5c. The rightmost /1 range still contains 3 samples, and since n_{cidr} is 8 for that range, nothing will happen because the minimum sample count is not reached.

By the time t_2, the leftmost /2 range meets its n_{cidr} requirement and is uniformly colored blue ($q = 1.0$), indicating a single dominant ingress point. It is thus assigned this ingress point. The other /2 range, now with sufficient samples but multiple ingress points, is further divided.

By the time t_3, we see another range being assigned a single ingress point (red). The process continues, with further splits and assignments, until either a unique classification is made or the maximum CIDR mask size ($cidr_{max}$) is reached. In our operational setting, $cidr_{max}$ is set to /28 because the collaborating CDN maps its geolocation-distributed data centers to /28 subnets, and the ISP needs to recognize the ingress point changes.

This iterative aggregation, splitting, and classification process efficiently narrows down the ingress points for vast IP ranges, using real-time traffic data to reflect current network conditions.

IPD algorithm details. Our *IPD* algorithm, shown in Algorithm 1, operates in two stages, executed in parallel threads: *i)* The first stage identifies *all* ingress points for each range by processing flow data from all border routers. Each source IP is masked to $cidr_{max}$ and inserted into a binary tree data structure, one for IPv4 and one for IPv6 (lines 1-4). *ii)* The second stage identifies the prevalent ingress point one per prefix (line 5-19). To do so, it iterates through all ranges every t seconds, aggregating them into *IPD* prefixes.

After each t-second cycle, the algorithm checks and updates the *IPD* ranges, removing expired data and maintaining state only for ranges lacking a definitive ingress point. This state includes the sample counter, the respective ingress, and the last timestamp. Once a prevalent ingress is found, all state is removed for efficiency reasons, and only the total number of samples, the counters for the respective ingresses, and the last timestamp are retained. Source IP information older than e seconds (default: 120s) is removed from the range. For already classified ranges, it is still being determined when to remove state. We thus employ a decay function (see Table 1) that quickly reduces the counter values. In our implementation, we use an exponential decay function to quickly reduce the counters. This ensures that ranges are quickly removed from classification when no new traffic is received.

Internally, we keep state for all yet unclassified ranges. *IPD* checks if these ranges meet the minimum sample threshold (n_{cidr}) and then either assigns a prevailing ingress or splits the range for finer classification. Adjacent ranges may also be joined if they share the same ingress and meet sample count requirements. Special handling is needed for evenly distributed traffic across multiple router

[1]How robust is *IPD* against IP spoofing? IPD is designed to detect high-traffic prefixes' entry points with high granularity. Attacking those would require to generate substantial traffic volumes (to be noticeable, the volume needs to be $> q$ for the attacked prefix and substantially larger to become the dominant ingress point). This is unrealistic in practice and can be easily detected through other means, such as massive provisioning of high-capacity directly attached links.

Algorithm 1: *IPD* parameters in red (see Tab.1)

```
/* thread 1 (Stage 1)                                    */
while True do
    read Netflow row
    mask src_ip to $cidr_{max}$
    add ts, masked_src_ip, ingress_link to range
/* thread 2 (Stage 2)                                    */
Every $t$ seconds:
for current_range in all_ranges do
    remove/decrease expired ranges older than $e$ sec
    // handle unclassified ranges
    if check sample counts ($s_{ipcount} \geq n_{cidr}$) then
        if single ingress prevalent ($s_{ingress} \geq q$) then
            set ingress_link to current_range
        else
            if $cidr_{max}$ not yet reached ($s_{cidr} < cidr_{max}$)
              then
                split current_range
            else
                try to join
    // handle already classified ranges
    if prevalent ingress still valid ($s_{ingress} \geq q$) then
        try to join
    else
        drop current_range
```

interfaces, where they are bundled as a single logical ingress (called *bundles*).

Finally, it is determined whether prevalent ranges remain valid. Invalidated and expired ranges are dropped, ensuring *IPD* adapts to changing traffic patterns. Due to the single-threaded nature of this stage, the runtime scales linearly with the number of ranges, underscoring the necessity of efficient range management to maintain prompt *IPD* operations.

Parametrization. To identify an optimal parameter set for *IPD*, we performed a systematic parameter study using a full factorial design on 25 hours of Netflow data (§ 4). In this study, we evaluated 308 different parameter combinations subject to *IPD* accuracy, stability duration, and resource usage. Our study shows that the parameter configurations have no significant effect on *IPD*'s accuracy—i.e., *IPD* cannot perform worse when configured suboptimally. The reason is that *IPD* is not a learning system, rather it performs a traffic-based partitioning of the IP address space. Thus, a suboptimal configuration will waste memory or CPU resources but not result in a suboptimal accuracy. Yet, the parameters q and $cidr_{max}$ affect stability and resource consumption (both *IPD* iteration time and average memory usage increase exponentially with higher $cidr_{max}$ values). For details on this study, see Appendix A. The optimal parameters used in our production deployment are shown in Table 1.

Open Source Release. We release a prototypical *IPD* implementation at [24]. To quickly experiment with *IPD*, we further release a Mini IPD environment [25] that uses the Mini Internet [14] and includes an example ISP scenario—ready for research and teaching.

Table 1: Default *IPD* parameters.

Parameter	Default	Meaning
$cidr_{max}$	/28, /48	max. *IPD* prefix length
$n_{cidr}factor$	64, 24	minimal sample factor $n_{cidr} = n_{cidr}factor * \sqrt{2^{(32-s_{cidr})}}$
q	0.95	error margin
t	60	time bucket length
e	120	expiration time
$decay$	$1-\frac{0.9}{(\frac{age}{t})+1}$	factor to reduce outdated *IPD* ranges

4 DATA SET & ETHICS

This section details the datasets used to evaluate the *IPD* deployment at a major tier-1 ISP over six years.

Netflow traffic traces. The ISP captured 25 hours of flow data for validation at all border routers. It is anonymized by including only ingress points, source IPs (aggregated to /28 prefixes for privacy), and timestamps. This dataset, stripped of payload, comprises 48 billion flows (average 32 million per minute) and adheres to privacy standards by aggregating IPs to /28 to make users unidentifiable. Data capturing complies with local legal regulations, follows ISP practices for network debugging, and was stored on-premise.

***IPD* output data.** We obtain six years of raw *IPD* output data (see example data in Appendix B) from the ISP. It covers mapped and non-mapped prefixes and their ingress points at 5 minute granularity. This extensive dataset (2.5T compressed) enables to study the *IPD*'s deployment experience.

BGP tables. To study path symmetry, we further obtain periodic BGP table dumps from the same period as the *IPD* data. These are supplemented with link classifications (e.g., PNI) and mappings of routers and links to connected ASes.

The datasets focus solely on topological data, excluding traffic or user-specific information, thereby posing no privacy concerns. Our handling of this data strictly adheres to ethical standards.

5 SIX YEARS OF *IPD* AT A TIER-1 ISP

Our *IPD* algorithm has been in operation at a major tier-1 ISP for six years. In this section, we use data obtained from this deployment to evaluate the *IPD* approach in an ISP setting. Specifically, we focus on three main aspects: *i)* How well does the *IPD* approach work in practice?, *ii)* How dynamic are *IPD* ranges and ingress points, and *iii)* What operational insights can we obtain, focusing on path asymmetry and peering violations.

5.1 (How Well) Works *IPD* in Practice?

Approach. We validate the *IPD* algorithm's accuracy by contrasting its determined ingress points with ground truth data derived from 25 hours of sampled Netflow traffic traces. These traces were collected from all ingress routers at the ISP's premise (see § 4). The

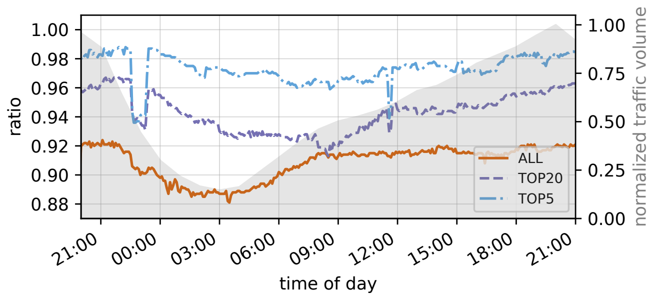
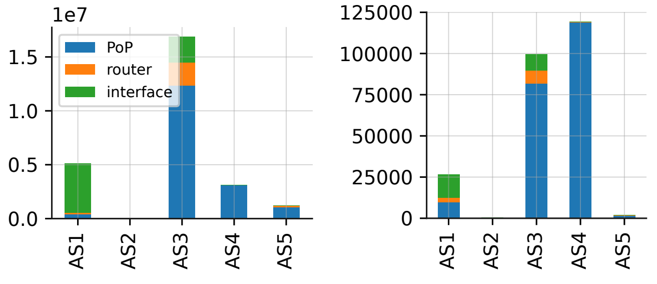


Figure 6: *IPD* accuracy (vs. ground truth Netflow data)



(a) # misses **(b) # distinct source IPs**

Figure 7: *IPD* misclassifications for TOP5 ASes

validation process involves a three-step approach. First, we create a Longest Prefix Match (LPM) lookup table from the *IPD* output that maps each *IPD* prefix to its corresponding ingress router and interface. Second, we process the flow traces and compare the actual ingress router and interface with the *IPD* output by using the LPM lookup table. Finally, we calculate the ratio of correctly classified flows (i.e., with matching ingress point) relative to all flows in a time bin. Since the *IPD* prediction results are provided in 5-minute bins, we recompute the lookup table after every 5-minute bin to ensure we are using the latest available information. By doing so, we can then compare the output of the *IPD* prediction to the same flow data that was used as the original input to the *IPD* algorithm.

A key focus for traffic engineering is on the prefixes associated with ASes that account for a significant share of the traffic. To facilitate these analyses, we have identified the top 5 ASes (referred to as "TOP5") that constitute 52% of the total volume, as well as the top 20 ASes (referred to as "TOP20") that account for 80% of the traffic. The term "ALL" denotes the complete set of flows.

5.1.1 IPD achieves high classification accuracy. In Figure 6, we show the accuracy of the *IPD* algorithm for TOP5 (blue line), TOP20 (purple line), and the complete traffic (orange line). The maximum normalized traffic volume (diurnal pattern) is shown as a gray shade. Looking at the unfiltered set of flows (ALL), on average 91% of the flows are classified correctly. The accuracy increases to 94% on average when looking at TOP20, respectively to 97.4% for TOP5. This shows that *IPD* is able to accurately classify the ingress points for prefixes that contribute the major portions of the Internet traffic. Since these are the prefixes that matter for traffic engineering, *IPD* works very well from an ISP perspective.

5.1.2 Few IPD Misclassifications. This raises the question of why *IPD* does not correctly identify 100% of the flows. There are a few unavoidable reasons for misclassifications, including sudden bursts of traffic, noise, unbalanced load balancing, or the moment when prefixes shift from one ingress point to another. To study *IPD* misclassifications, we will focus on the TOP5 ASes. There are three main types of misses:

(1) Interface miss: Traffic enters through a different interface on the same router.
(2) Router miss: Traffic enters through another router within the same Point of Presence (PoP).
(3) PoP miss: Traffic enters at a different geolocation.

To study the prevalence of these three types, we show their frequency in Figure 7 as *(i)* absolute miss counts by type per AS (left plot) and *(ii)* the number of distinct source IPs (right plot). AS3 experiences the highest number of misses, with PoP misses being the dominant type—similar to AS4. In contrast, AS1 sees a higher number of interface misses.

By examining the misses over time (see the upper plot in Figure 8), we observe notable peaks in AS1 around 11 AM and 11 PM. These peaks correspond to the short drops in accuracy seen in Figure 6. AS3 and AS4, show distinct diurnal patterns, as shown in the lower plot of Figure 8. These patterns may be attributed to the user-server mapping functions of these CDNs, which respond to fluctuating on-demand traffic. In contrast, AS1 shows consistent misses caused by source IPs throughout the observed period. We will next discuss the reasons for the misses in these three ASes.

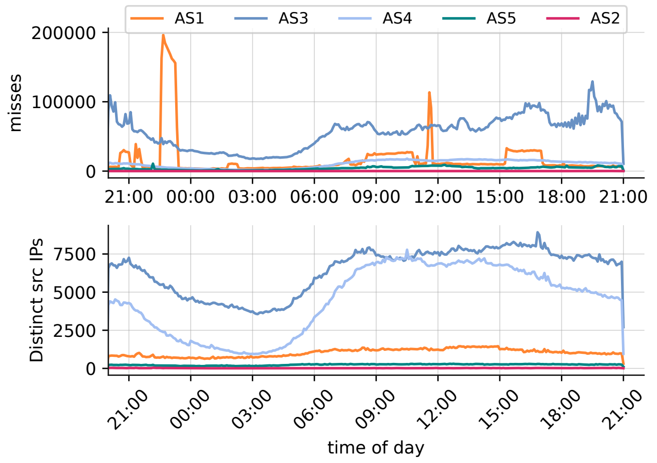


Figure 8: *IPD* misclassifications of the TOP5 ASes

Starting with AS1, we have identified small prefixes (/25 to /27) that are responsible for over 65% of all the misses for this AS. In these cases, *IPD* has classified a bundle (where multiple interfaces of the same router are logically mapped as one link in our algorithm) with two interfaces, and all the misses originate from two other interfaces. By interacting with the ISP, we identified that maintenance was performed on that router (e.g., replacement or upgrade) at these times, which is the cause for the observed misses. Since each prefix already has 400 million to 600 million samples, the confidence is accordingly high (indicating a stable link). The more than 70,000 misses that occur between 10:35 PM and 11:20 PM only slightly lower the confidence because there are consistently over 80,000 flows per minute enter the expected ingress. This highlights that *IPD* demonstrates a certain level of robustness against noise for high-traffic prefixes.

In AS4, three prefixes are also responsible for most of the misses. However, they cover relatively large address space with /12 to /15 network sizes this time. When the maximum normalized traffic curve of the entire AS is overlayed with the miss counts throughout the day, they almost perfectly align. The correlation coefficients for these three prefixes with the traffic range between 0.88 and 0.99. This confirms the assumption that these are artifacts of the user-server mappings by the CDN. These prefixes already contain between 1.1 billion and 2.2 billion samples, making the relatively few misclassifications negligible.

In the case of AS3, there isn't such a clear picture. Networks of various sizes generate all types of misses, with the majority clearly being PoP misses. Let's focus on the top three prefixes. The first prefix exclusively consists of PoP misses. Here, 1 million flows enter the network through a router in another country. Most likely, there is a misalignment in the CDN mapping that may result into a performance bottleneck as described later in § 5.8. Further, the number of errors follows a daily pattern that correlates with the AS's traffic (corr. coefficient of 0.84). The second prefix exclusively experiences router misses. Upon closer examination, it is precisely two routers at the same PoP that receive misses in roughly equal proportions. This is undoubtedly router-based load balancing, which is currently not recognized by *IPD*. The third prefix, on the other hand, contains exclusively interface misses, which can be argued similarly to AS1.
Takeaway. *IPD can accurately compute the ingress router and interface in practice. Overall, IPD is relatively robust against noise induced by traffic shifts or operational changes.*

5.2 Distribution of *IPD* Ranges

A crucial design requirement is partitioning the IP space into *IPD* ranges by traffic (refer to § 3.1). These ranges are expected to be independent of BGP prefix sizes, making static partitioning (e.g., as in [22]) suboptimal. We now demonstrate that *IPD* range sizes vary and are independent of BGP.

In Figure 9, we present the distribution of IPv4 *IPD* range sizes for the entire dataset (in orange) alongside BGP prefix sizes (in gray), and TOP5 and TOP20 subsets (in blue and purple, respectively). Initially, comparing the 'ALL' category with BGP reveals that both distributions are markedly different. Notably, a small number of *IPD* ranges are observed between /7 and /13, whereas relatively few BGP prefixes are larger than /29.

For clarity, our plots primarily focus on masks ranging from /14 to /28. Here, announcements of /24 prefixes in BGP constitute over 50% of the total. Prefixes from /20 to /23 each represent a share of 5% to 10%. In contrast, the *IPD* algorithm segments the IP address space into smaller parts, varying based on ingress links. For example, if 1.2.0.128/26 and 1.2.0.192/26 enter via the same ingress, they are aggregated into 1.2.0.128/25. Furthermore, some CDNs use internal addressing that goes down to /28, mapping users to servers. Limiting the *IPD* to a maximum CIDR of /24 would hinder its ability to classify such ranges, affecting CDN traffic steering (see, for instance, [28]). The TOP20 subset tends towards smaller networks. The TOP5 subset, part of TOP20, shows a distribution similar to 'ALL' but with a significantly larger proportion of /24 networks.

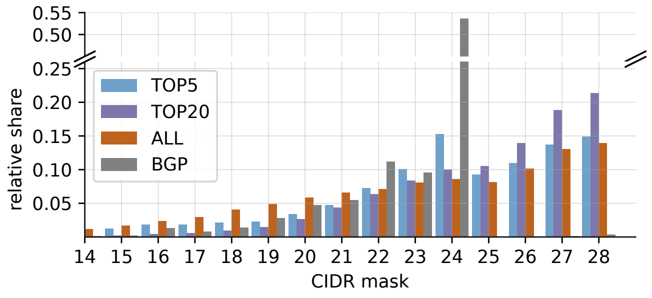


Figure 9: Distribution of *IPD* ranges

Takeaway. *The traffic-based partitioning of the IP address space leads to varying IPD range sizes that directly capture service dynamics (e.g., the granularity CDN server mappings). They are thus unrelated to BGP prefix size and should also not be kept static (e.g., /24 only) to enable CDN traffic steering.*

5.3 Traffic and Ingress Point Dynamics

Dynamic ingress point changes due to traffic engineering, CDN adjustments, and infrastructure upgrades necessitate frequent *IPD* updates for ISPs. Our temporal analysis from an ISP's standpoint highlights these fluctuations, particularly during peak traffic periods. As we will show, dynamics are time dependent: stability during peak traffic times largely differs from stability throughout the day.

While these are not directly actionable results, they intend to inform *IPD* users about what to expect from an *IPD* prefix/range. Network operators are used to understand routing tables, more specific prefixes, etc. Since the *IPD* is a traffic-based aggregation of the IP address space rather than a routing/allocation-based aggregation, it is relevant to highlight how the granularity of the prefixes follows traffic patterns.

5.3.1 Longitudinal Ingress Point Stability at Prime Time. We evaluate ingress point stability over extended periods. We examine if addresses that ingress at a particular link today will still ingress at the same link in one day, one week, one month, or one year from now. Note we explicitly refrain from comparing *IPD* ranges to avoid bias from the algorithm's dynamic aggregation adjustments. Instead, we directly compare the stability of addresses in the mapped space.
Approach. We study this longitudinal behavior by evaluating the stability of one timestamp t_1 to all following timestamps t_2 in a specified interval such as once a day. As time of day, we chose a high-traffic busy hour at 8 PM local time, which refers to the busiest period for this ISP in which the most traffic is being carried. Optimizing traffic at prime time can be a relevant objective from an operational perspective.

Our analysis is twofold. First, we evaluate how much address space from timestamp t_1 still exists at t_2. To accomplish this, we create an LPM trie with all prefixes from t_2 and looked up the addresses of each prefix that exists at t_1. We refer to addresses that exist both at t_1 and t_2 as *matching*. Second, we define addresses as *stable* if they exist at both timestamps t_1 and t_2 and ingress at the same link. We define addresses as *unstable*, when they ingress at different ingress links at both timestamps t_1 and t_2.

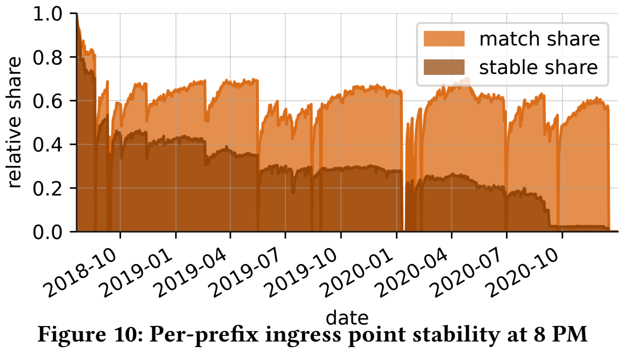


Figure 10: Per-prefix ingress point stability at 8 PM

Results. In Figure 10, we show the relative ratios of both *(i)* matching and *(ii)* stable addresses for various combinations of timestamps, t_1 to all subsequent t_2, as time series plots. For t_1, we selected the timestamp '2018-07-20 8 PM', and the interval is set to once a day. In Figure 10, we observe a consistent trend: the matching share drops significantly after a few weeks, reaching a different plateau. Further, the matching ratio decreases to approximately 60%. This implies that only 60% of the prefixes or portions thereof from t_1 are present in the corresponding t_2. The proportion of prefixes that remain at the same link initially drops, stabilizes at around 50%, and then steadily decreases over time, with hardly any prefixes from t_1 remaining on the same link after September 2020.

In the case of TOP20 and TOP5 (not shown), the relative share of matching prefixes is slightly higher. Similarly, the stable share declines and then maintains a relatively low and constant level of around 20%.

Takeaway. *By focusing our analysis on the comparison of a specific timestamp every day, we observe that, after just a few weeks, both matching and stability shares decrease significantly. This suggests that ingress points exhibit highly dynamic behavior over time, emphasizing the need for frequent IPD measurements to consistently maintain an accurate picture.*

5.3.2 *Ingress Point Stability Throughout the Day.*

The stability of ingress points at prime time does not mean that ingress points are also stable throughout the day. Traffic patterns change over the day (diurnal pattern [8]), and as a result, the output of the *IPD* is also expected to exhibit similar dynamics. Therefore, we will study the dynamics of ingress point mappings throughout the day. We will show that the number of *IPD* prefixes fluctuates substantially over the day while the number of classified IP addresses (i.e., the mapped address space) remains stable.

Approach. We aggregated the distribution of prefixes per mask (in other words: the network size) for all TOP5 ASes for the period from 2018-01-01 to 2020-12-31 at a representative day. We show the resulting distribution in Figure 11. The top plot shows the classified IP address space while the plot at the bottom shows the relative number of *IPD* prefixes. The y-axes of both plots are normalized to the maximum number of mapped IP addresses or *IPD* prefixes,

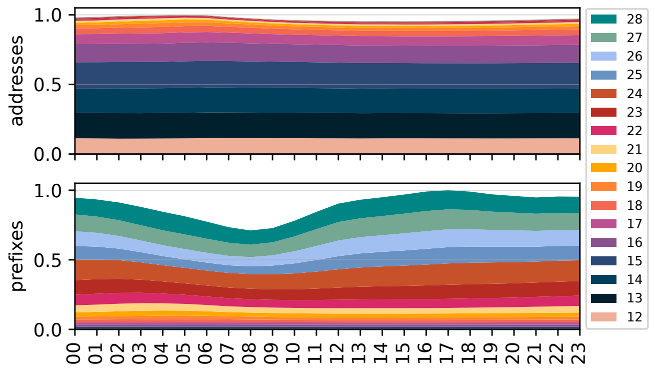


Figure 11: Distribution of network size aggregated by daytime for TOP5 ASes from 2018 to 2020

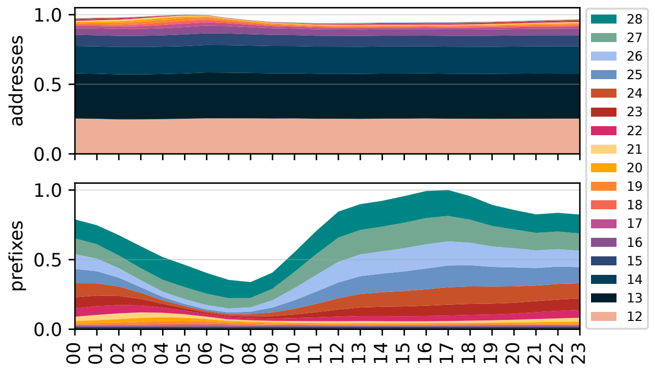


Figure 12: CDN Behavior: Network size distribution of AS4 over the period of 2018 to 2020

respectively. The colors in the plots represent the network size based on the network mask (see the legend on the right side).

Results. While the address space (number of prefixes weighted by mask) remains relatively stable, with a slight decrease during the afternoon, the number of prefixes undergoes notable fluctuations. It decreases to 70% by 6 AM - 7 AM, subsequently experiences a slight increase, peaking at 4 PM, and maintains this level throughout the evening. This variability in the number of prefixes can be attributed to the dynamics of ingress traffic. During periods of lower traffic, larger *IPD* ranges appear to emerge because sibling CIDR ranges receive traffic from the same ingress point, and there is less overall traffic within the network that requires classification. Furthermore, demand-based traffic engineering may result in prefixes being completely excluded from the *IPD* dataset. This occurs when a request from the CDN is mapped to a different server, causing specific prefixes to fall out of the classification of *IPD*.

Takeaway. *While the size of the mapped address space remains relatively stable throughout the day, the number of IPD prefixes—and thereby their granularity—varies substantially over the day. This is a consequence of merging prefixes in low traffic periods at night time and early morning and splitting into more specific ones after peak traffic at prime time.*

5.3.3 *IPD Prefix Dynamics in Detail.*

There are various reasons for ingress point or *IPD* range changes. To take a detailed look at these dynamics, we next examine the address space of AS4 (which is a CDN) that we study in § 5.1.2. CDNs use a network of distributed servers to deliver content faster by redirecting users to CDN servers in their proximity [10, 20, 27]. This dynamic redirection can affect CDN traffic entering the ISPs network from different ingress points. As we will discuss in § 5.8, one motivation for ISPs to run *IPD* is to implement collaborative CDN-ISP approaches to optimize the incoming traffic.

We now study the effect of CDN dynamics on the *IPD*. Similar to the previous section, we show the size of the mapped address space (top) and the number of *IPD* prefixes (bottom) for the address space of the selected CDN in Figure 12. While the size of the mapped address space remains relatively stable throughout the day, similar to what we observed for TOP5 ASes in Figure 11, we now notice a clear diurnal pattern when focusing on the number of prefixes, as indicated in the lower plot. These findings align with our analysis, where we investigated the reasons for *IPD* classification misses, as shown in the lower plot of Figure 8. After reaching a peak value at 4 PM, the number of prefixes decreases to less than 40% by 6 AM. Most range sizes are consolidated during this time, meaning that /26 to /22 networks are aggregated into larger networks. In the upper plot, it is evident that the number of addresses from /13 networks increases. This is likely an artifact of a demand-based mapping strategy employed by AS4. We also examined other ASes within the TOP5 group. We also observed fluctuations here, but they were less pronounced and occurred at various times throughout the day.

5.3.4 Reaction to Changes. To study how the classification reacts to change (e.g., how *IPD* captures traffic shifts), we now focus on a specific IP range within the AS. This discussion also aims to exemplify parts of the *IPD* operation on a practical example. Figure 13 shows various classified *IPD* ranges within a /23 CIDR prefix. Different colors represent distinct ingress points, including their ingress routers, links, and countries. The opacity of the colors indicates whether the ingress has been classified for that time—full opacity means it has, while reduced opacity suggests that the algorithm has been monitoring the range, but the minimum flow count or confidence level has not yet been reached. Since the /23 range has two ingress points for most of the time, the range is split as needed. As a result, traffic from 'x.y.196.0/25' and 'x.y.197.0/24' both consistently enters through the same ingress point until July 14, 2020. Afterward, the interface changes. Notably, the former range experiences occasional gaps in classification due to lower traffic, explaining the larger discontinuities. The ingress point change on July 14, 2020, is particularly interesting because it coincides with an ISP router maintenance event. During this event, the router device was either replaced or upgraded. The subnet 'x.y.196.128/26', which is between the two ranges above, enters through a different ingress point. Starting from July 26, 2020, the range is entirely excluded from *IPD* classification, and classified again as aggregated prefix 'x.y.196.0/23' on July 29, 2020, through the red ingress.

Figure 14 provides a detailed view of the 'x.y.197.0/24' range. Each color still corresponds to a unique ingress point, with opacity indicating the classification state. The lower graph displays counters for each ingress point, with a gray dotted line depicting the total counter. The upper graph represents the confidence as a blue line and minimum classification values as a red dashed line. The sample

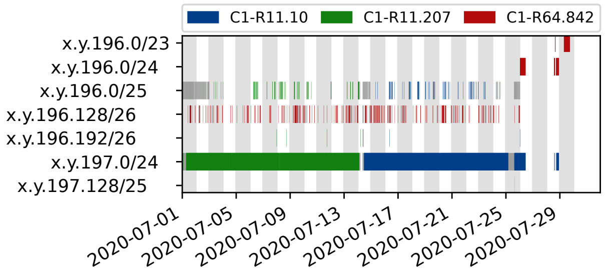


Figure 13: Classification status of different *IPD* ranges. Colors represent different ingress points, including countries (Ci), routers (Ri), and links. Full opacity indicates classified ranges, while low opacity indicates ranges that are not classified yet.

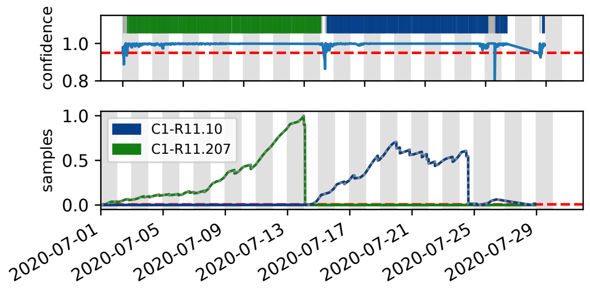


Figure 14: Detailed view of prefix x.y.197.0/24. Clearly visible: The ingress change at 2020-07-14.

counter constantly increases until the maintenance event, with the confidence continuously maintained. During the maintenance event, the range is excluded from classification but is reclassified at a different interface shortly after.

Takeaway. *IPD can quickly detect ingress point changes, enabling it to maintain an accurate state.*

5.4 Characterizing Elephant Ranges

Some *IPD* ranges have very large sample counts. This can be either due to *(i)* a persistent ingress over a long period of time or *(ii)* a significant amount of traffic entering the same ingress point. Following the terminology of Curtis and Benson [2, 5], we refer to these as *elephant ranges.*

We analyze the top 1% *IPD* ranges with the highest sample counters (7,818 distinct ranges). 33.4% of those are PNI links, 10.9% (26.3%) belong to the TOP5 (20) ASNs, respectively. The range sizes roughly follow a normal distribution, with /17 ranges being the most prevalent (not shown). It is worth noting that most ranges do not originate from the address spaces of the TOP5 or 20 ASNs.

Stable for long periods. Figure 15 compares the stability of elephant to all ranges (denoted as 'ALL baseline'). To compute the duration of stable phases, we tracked how long the sample counter monotonically increased for each range. While 60% of all ranges are stable for an hour or less, the stability of elephant ranges is typically in the order of months.

We next analyze the number of new flows entering the network for each time bucket and prefix. On average, we observed that there

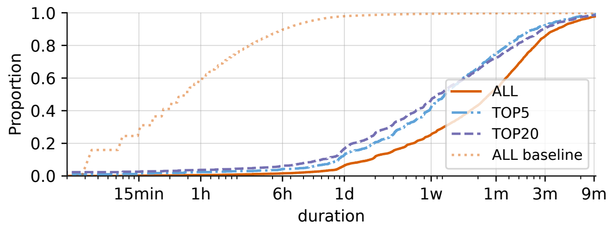


Figure 15: Stability of elephant ranges

are 2,000 new flows for the TOP20 prefixes and up to 10,000 for ALL prefixes within each time interval. This leads us to conclude that the high sample counts are likely a result of consistently stable ingress points, rather than sporadic, large traffic bursts.

Takeaway. *IPD ranges with high counter values exist. They could easily be (mis-)interpreted as high traffic volumes, however, they more often result from long stability phases.*

5.5 Path Asymmetry: *IPD* vs. BGP

Inferring ingress points is in practice sometimes simplified by taking easy to obtain BGP feeds and assuming path symmetry. Having the *IPD* deployed, we can evaluate this practice and shed light into the path (a)symmetry from the perspective of this tier-1 ISP—showing that BGP should not be used.

Approach. We compare *IPD* ingress routers with egress routers from historical BGP table dumps for each timestamp. Given that *IPD* traffic-derived ranges and BGP prefixes don't align, we categorize them into three scenarios: *a)* exact matches, *b) IPD* ranges more specific than BGP prefixes, and *c) IPD* ranges less specific than BGP prefixes.

BGP and *IPD* prefix correlation. Predominantly, *IPD* ranges are more specific than BGP prefixes (91%), highlighting that parts of a single BGP prefix can have different ingress points. Thus, even if we (unrealistically) assume path symmetry, using BGP for ingress point detection would be too coarse to capture ingress points. Exact matches are rare (1%), underscoring the distinct nature of *IPD* ranges from BGP data. *IPD* ranges being less specific than BGP prefixes (8%) shows cases in which neighboring BGP prefixes share ingress points and can thus be joined into larger *IPD* ranges.

Takeaway. *Most IPD ranges (91%) are more specific than BGP ranges, which highlight diverse ingress pattern within a single BGP prefix.*

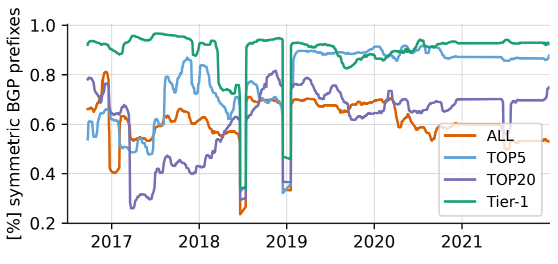


Figure 16: Traffic symmetry ratios over time

Path asymmetry. We evaluate path asymmetry by comparing ingress (IPD-derived) and egress (BGP-derived) points for all prefixes, and specifically for TOP20, TOP5, and tier-1 ASes from 2017 to 2022. Figure 16 shows their symmetry ratios. Our analysis reveals that, on average, 62% symmetry is observed across all prefixes, with TOP20 ASes at around 61%, and TOP5 ASes significantly higher at 77%. Notably, tier-1 ASes achieve a remarkable average symmetry of 91%.

These symmetry rates surpass those reported in prior research [11, 12], aligning instead with recent findings at IXPs [3], where symmetry rates ranged from 79-88% for prefixes (66-86% at the AS-level) [3]. Our approach is distinct as it assesses if ingress and egress routers coincide, diverging from other methodologies that rely on active path measurements.

Takeaway. *We observe a substantial amount of path asymmetry, highlighting that egress and ingress points differ. Due to the asymmetry, BGP cannot be used to predict ingress points.*

5.6 Case Study: Identifying Possible Peering Agreement Violations

This section explores another application of *IPD*: identifying potential violations of settlement-free peering agreements, where it's expected that traffic from a peering network enters directly via peering links, not through third parties.

Approach. We monitor the ingress of prefixes of 16 tier-1 ISPs (from daily BGP dumps), to check if traffic from these peers bypasses direct peering links. Traffic from a tier-1 AS entering our ISP network through non-peering links may indicate possible peering agreement violations.

Results. Between March 2018 and December 2021, about 9% of tier-1 ISP prefixes entered our ISP indirectly, possibly indicating peering agreement violations. Such situations can strain peering links, leading to congestion and degraded performance. Figure 17 illustrates the absolute frequency of these instances, categorized by ISP. An upward trend in potential violations is apparent, with the number of such instances increasing by 50% from September 2019 and doubling by 2020 across all tier-1 ISPs studied.

It is important to note that we cannot confirm violations without access to the peering agreements. Yet such traffic patterns are generally unexpected among peering partners.

5.7 Operational Deployment

IPD is deployed at the tier-1 ISP for the past six years. Next, we briefly discuss the deployment setup and the resource requirements.

The tier-1 ISP effectively operates the *IPD* on a single 48-core server with 500 GB of RAM. This machine can handle the incoming flow records from all $\approx 3,000$ routers. Recent data captured at the deployment server reveals that, on average, the machine receives and processes live 300 billion flow records per day and 4 million flow records per second on average. During peak hours, the server processes 22 billion flow records/hour and 6.5 million flow records/second.

To handle this load, the server has an average load of 30 cores, which results from the processes that handle incoming flow data and a single-core process that executes the central part of the *IPD*:

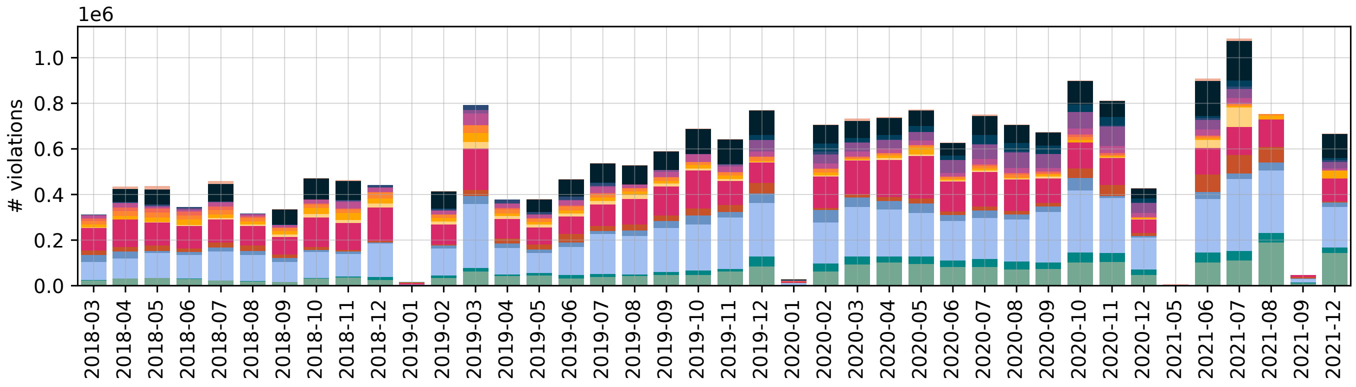


Figure 17: Violations of tier-1 peering agreements over time, with each color representing different tier-1 ISPs.

mapping the ingressing flows to IPD ranges. The typical total memory consumption of all processes is 120 GB (120 MB per flow reader process, the rest for the primary *IPD* process). Thus, the 48-core server with 500 GB of total RAM is far from fully loaded. This single server runs the *IPD* for the entire tier-1 ISP.

Note that the *IPD* parametrization impacts the memory consumption, as we show in Appendix A. Here, setting a more specific $cidr_{max}$ than /28 for IPv4 (/48 for IPv6) as in our case, e.g., /29 or /30, will increase the number of *IPD* ranges and thus the amount of state needs to be managed by the *IPD* process and hence will increase the memory consumption. Given the current traffic dynamics of hyperscalers (e.g., CDNs), such a change is unnecessary.

These deployment statistics show that resource requirements for running *IPD* at a major ISP are relatively moderate: a single server.

5.8 Operational Experience

IPD enables ISPs to better run their networks by identifying where traffic enters their networks. The deployment of *IPD* at the studied tier-1 ISP gave further insights into operational experiences and considerations, which we will briefly discuss in this section. At the studied tier-1 ISP, *IPD* enabled the ISP to better debug performance problems and to run data-driven products that optimize traffic engineering.

"Why is service X slow at home in only one city of an ISP's network?" The studied tier-1 ISP also operates a residential access network with ADSL and FTTH access lines. In one situation, the *IPD* enabled the ISP to quickly identify a performance problem, in which a major Internet service was slow in a certain city-level region—yet it was slow only for FTTH customers, not for ADSL customers in the same city. The *IPD* revealed a CDN-based mapping problem. Here, the traffic to prefixes assigned to FTTH customers entered the ISPs network in a different, further away country than the traffic to the ADSL customers in the very same city. The reason was that the CDN mapped them differently and selected different data centers. The *IPD* identified this issue and enabled the ISP to resolve the mapping problem with the CDN. Resolving it would have been possible without *IPD*, but it is way more cumbersome and resource-intensive. This highlights a major strength of the *IPD* for ISPs: i.e., to quickly gather enough debugging information to be able to talk to interconnected networks.

IPD further helps to display non-optimal routes, e.g., CDN traffic that enters the ISPs' network via non-direct links. One example are networks that hand-over traffic in other countries via other ASes while having a local presence and direct links (e.g., direct, private connections between two ASes, referred to as Private Network Interconnect) to the ISP. Such events can be the result of overflow events [4]. They are sometimes detrimental to network performance. Yet, *IPD* can easily reveal their existence, e.g., via dashboards.

ISP-CDN collaboration. The use of the *IPD* enabled the ISP to create data-driven products that can be used for traffic engineering. The studied ISP uses the *IPD* as one component to build a platform [28] that enables automated cooperation between the ISP and CDNs to jointly optimize traffic engineering. This problem is referred to as hyper-giant traffic steering and aims to jointly solve two complex operational problems: *i)* performing inbound traffic engineering by the ISP and *ii)* mapping user to servers by the CDN.

When does IPD not work well? The *IPD* primarily aims to optimize ingress traffic engineering (e.g., CDN-ISP traffic steering, see above) and identify problems requiring network operators to determine how traffic enters their network. While this sounds simple, it is a problem that even major ISPs can struggle to solve, which thus resulted in us developing *IPD*. Consequently, any problem that does not fall into this category is beyond the scope of what we aim *IPD* to be used for.

Yet, we omitted a specific aspect in the design of *IPD*, which can result in operational issues: router level load balancing. To simplify the design of *IPD*, we have intentionally not considered router-level load balancing over multiple routers.

Yet router-level load balancing can occur in practice—in our case, it is just too infrequent to be worth considering. An operation issue occurred once in the deployment history, where a directly connected hypergiant balanced traffic over two routers. This resulted in *IPD* being unable to classify the corresponding prefixes of this hypergiant accurately. We have discussed if this problem should be solved and decided against accounting for router-level load balancing.

The reason is that detecting this kind of load balancing requires tracking both the destination IPs and the source IPs. However, keeping all (source, destination) IP pairs results in a quadratic additional state complexity and, thus, memory/RAM utilization. Since router-level load balancing by the networks connected to this tier-1

ISP rarely occurs, we decided that this substantial additional RAM usage is not worth the benefit of adequately handling the very few cases of load balancing (which can also be solved by asking interconnected networks to change their configuration).

This design decision is based on our experience and works for the ISP where *IPD* is deployed. However, other network operators might have different requirements, and thus, an extension of *IPD* in future work to handle router-level load balancing might be of interest. For example, by tracking the (source, destination) IP address pairs.

6 RELATED WORK

This work provides the first approach for detecting traffic ingress points at ISPs without using BGP.

Ingress Point Detection. Closest to and parallel to our work is TIPSY [22], which focuses on the traffic properties of major cloud provider, whereas *IPD* is tailored to the traffic dynamics of ISPs. TIPSY aims to statistically model ingress traffic volumes and points for each /24 prefix, while *IPD* aims to capture the traffic dynamics of hyperscalers such as clouds or CDNs at dynamic prefix sizes (see § 5.2). TIPSY's ingress traffic volume mapping approach's primary motivation is to enable congestion management at a major cloud provider. It enables the congestion manager to predict the effect of shifting traffic by selective BGP withdrawals to mitigate congestion, for prefixes observed in a training period. In contrast, *IPD* assumes the ingressing IP space to be highly dynamic and thus always maps the entire address space without requiring a learning period or input from BGP. Further, the ingress points at ISPs exhibit high dynamism, as we have observed in § 2 and § 5.3. Therefore, the dynamic approach taken by *IPD* is optimized to capture these dynamics, which is a major difference from TIPSY. Therefore, TIPSY and *IPD* are two distinct solutions for ingress point inference, optimized for hyperscalers and ISPs, respectively.

Traffic engineering. ISPs generally rely on traffic engineering [23] approaches to steer the incoming or outgoing traffic. *IPD* supports these approaches by informing the ISP *where* traffic enters the network, enabling to decide if traffic steering is required. Classical means of traffic engineering use BGP to alter paths [29]. While ingress traffic engineering practices can be always overwritten by the sending network, egress traffic engineering can be precisely controlled by BGP (see e.g., how to control the outbound traffic of stub ASes with BGP [33]). As BGP does not incorporate link loads, it cannot balance or control link load. To address this limitation, recent traffic engineering approaches utilize network programmability to enable precise link load management. Examples include software-defined WANs such as B4 [16], SWAN [15], Espresso [35], and Edge Fabric [30]. In contrast to these new centralized traffic engineering approaches, earlier approaches used distributed algorithms to control link load [19] or to manage path congestion [7]. Since egress traffic engineering by CDNs can be detrimental to ISP performance, recent works proposed approaches for joint traffic engineering and content placement by both the CDN and the ISP [1, 6, 9, 13, 17, 31], giving the CDN provider more control over the end-to-end path [1], or using a centralized infrastructure or parts of it [21, 34]. As discussed in § 5.8, *IPD* enables fine-granular traffic steering in such settings. In this regard, *IPD* adds a valuable tool to the ISPs toolbox to enable more informed and fine-granular traffic engineering.

7 CONCLUSION

We show that ISP ingress traffic is highly dynamic with frequent ingress changes—optimal traffic management requires an online algorithm to infer ingress points at minute granularity. Thus, we introduce *IPD*, an online algorithm that solves a complex measurement problem by analyzing (large) streams of continuous flow data from *all* border routers. Its key idea is to partition the entire IP address space into segments sharing the same ingress point. Unlike prior work, it is agnostic to BGP, does not assume static prefix sizes, and quickly adapts to ingress changes—that frequently occur at (eyeball) ISPs, e.g., due to CDNs. *IPD* is highly accurate, with a classification accuracy of 95 % for identifying the correct ingress router and interface for prefixes that carry 80 % of the total ingress traffic—those that ISPs aim to optimize for.

IPD has been deployed at a major tier-1 ISP for six years, during which we have gained valuable insights in its operation. The deployment experience shows that the approach can scale up to a large international network handling high traffic levels and keep up running for years. In our setting, *IPD* is running in *real-time* on a single commodity server to continuously infer ingress points from hundreds of border routers at multi-digit Tbit/s traffic levels. This highlights that the *IPD* approach only has moderate hardware requirements, even when run at a big network. While the presented *IPD* satisfies the needs set in deployment by the tier-1 ISP, enhancements are possible to suit needs in other deployments. A potential enhancement would be incorporating a router-based load balancing detection, which we did not include to keep the approach simple as we found no empirical evidence that it would improve performance in our setting.

The *IPD* enhances network operation by enabling ISPs to identify ingress traffic related issues (e.g., performance bottlenecks or misconfigured CDN-based mappings) but also to launch enhanced traffic engineering approaches such as CDN-ISP collaboration. Beyond operation, *IPD* can be a useful research tool since it enables studying certain aspects of Internet traffic such as traffic dynamics or routing asymmetries. From our deployment experience, the biggest benefit of the *IPD* is not technical, but rather enabling ISPs to easily gather sufficient information to talk to other networks about fixing issues (e.g., CDN-related mapping problems).

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Jia Wang for their constructive comments. This work received funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of SPP 2378 (Resilient Worlds) project ReNO (grant number 511099228).

REFERENCES

[1] Mostafa H. Ammar, Ellen Witte Zegura, and Yimeng Zhao. 2017. A Vision for Zero-Hop Networking (ZeN). In *IEEE International Conference on Distributed Computing Systems (ICDCS)*.
[2] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *ACM CoNEXT*.
[3] Leandro M. Bertholdo, Sandro L. A. Ferreira, João M. Ceron, Lisandro Zambenedetti Granville, Ralph Holz, and Roland van Rijswijk-Deij. 2022. On the

Asymmetry of Internet eXchange Points -Why Should IXPs and CDNs Care?. In *International Conference on Network and Service Management (CNSM).*

[4] Jeremias Blendin, Fabrice Bendfeldt, Ingmar Poese, Boris Koldehofe, and Oliver Hohlfeld. 2018. Dissecting Apple's Meta-CDN during an IOS Update. In *ACM Internet Measurement Conference (IMC).*

[5] Andrew R. Curtis, Wonho Kim, and Praveen Yalagandula. 2011. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *IEEE INFOCOM.*

[6] Khaled Diab and Mohamed Hefeeda. 2019. Joint Content Distribution and Traffic Engineering of Adaptive Videos in Telco-CDNs. In *IEEE INFOCOM.*

[7] Anwar Elwalid, Cheng Jin, Steven Low, and Indra Widjaja. 2002. MATE: multipath adaptive traffic engineering. *Computer Networks* 40, 6 (2002).

[8] S. Floyd and V. Paxson. 2001. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking* 9, 4 (2001).

[9] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. 2013. Pushing CDN-ISP Collaboration to the Limit. In *ACM SIGCOMM.*

[10] Michael J. Freedman. 2010. Experiences with CoralCDN: A Five-Year Operational View. In *USENIX NSDI.*

[11] Yihua He, M. Faloutsos, and S. Krishnamurthy. 2004. Quantifying routing asymmetry in the Internet at the AS level. In *IEEE GLOBECOM.*

[12] Yihua He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker. 2005. On routing asymmetry in the Internet. In *IEEE GLOBECOM.*

[13] Nicolas Herbaut, Daniel Negru, Yiping Chen, Pantelis A. Frangoudis, and Adlen Ksentini. 2016. Content Delivery Networks as a Virtual Network Function: A Win-Win ISP-CDN Collaboration. In *IEEE GLOBECOM.*

[14] Thomas Holterbach, Tobias Bühler, Tino Rellstab, and Laurent Vanbever. 2020. An Open Platform to Teach How the Internet Practically Works. *SIGCOMM Comput. Commun. Rev.* (2020).

[15] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving High Utilization with Software-Driven WAN. In *ACM SIGCOMM.*

[16] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined Wan. In *ACM SIGCOMM.*

[17] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. 2009. Cooperative Content Distribution and Traffic Engineering in an ISP Network. In *ACM SIGMETRICS.*

[18] Frank J. Massey Jr. 1951. The Kolmogorov-Smirnov Test for Goodness of Fit. *J. Amer. Statist. Assoc.* 46, 253 (1951).

[19] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. 2005. Walking the Tightrope: Responsive yet Stable Traffic Engineering. In *ACM SIGCOMM.*

[20] Tom Leighton. 2009. Improving Performance on the Internet. *Commun. ACM* 52, 2 (2009).

[21] Hongqiang Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitu Padhye, and Ming Zhang. 2015. *Efficiently delivering online services over integrated infrastructure.* Technical Report.

[22] Michael Markovitch, Sharad Agarwal, Rodrigo Fonseca, Ryan Beckett, Chuanji Zhang, Irena Atov, and Somesh Chaturmohta. 2022. TIPSY: predicting where traffic will ingress a WAN. In *ACM SIGCOMM.*

[23] Jim McManus, Joseph Malcolm, Michael D. O'Dell, Daniel O. Awduche, and Johnson Agogbua. 1999. Requirements for Traffic Engineering Over MPLS. RFC 2702.

[24] Stefan Mehner. 2024. *Example Ingress Point Detection Implementation.* https://github.com/smehner1/ipd

[25] Stefan Mehner and Max Bergmann. 2024. *Mini IPD.* https://github.com/smehner1/mini-ipd

[26] Douglas C Montgomery. 2017. *Design and analysis of experiments.* John wiley & sons.

[27] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. 2010. The Akamai network: a platform for high-performance internet applications. In *ACM SIGOPS.*

[28] Enric Pujol, Ingmar Poese, Johannes Zerwas, Georgios Smaragdakis, and Anja Feldmann. 2019. Steering hyper-giants' traffic at scale. In *ACM CoNEXT.*

[29] B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig. 2003. Interdomain traffic engineering with BGP. *IEEE Communications Magazine* 41, 5 (2003).

[30] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *ACM SIGCOMM.*

[31] Abhigyan Sharma, Arun Venkataramani, and Ramesh K. Sitaraman. 2013. Distributing Content Simplifies ISP Traffic Engineering. In *ACM SIGMETRICS.*

[32] Barbara G Tabachnick and Linda S Fidell. 2007. *Experimental designs using ANOVA.* Vol. 724. Thomson/Brooks/Cole Belmont, CA.

[33] Steve Uhlig and Olivier Bonaventure. 2004. Designing BGP-Based Outbound Traffic Engineering Techniques for Stub ASes. In *ACM SIGCOMM.*

[34] Matthias Wichtlhuber, Robert Reinecke, and David Hausheer. 2015. An SDN-Based CDN/ISP Collaboration Architecture for Managing High-Volume Flows. *IEEE Transactions on Network and Service Management* (2015).

[35] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *ACM SIGCOMM.*

Table 2: The parameter study applied a full factorial design using these factors and levels.

factor	level(s)
t	[60]
e	[120]
q	[0.501, 0.7, 0.8, 0.95, 0.99]
$n_{cidr}factor_4$	[32, 48, 64, 80]
$n_{cidr}factor_6$	[12, 18, 24, 30]
$cidr_{max4}$	[20, 21, 22, 23, 24, 25, 26, 27, 28]
$cidr_{max6}$	[32, 34, 36, 38, 40, 42, 44, 46, 48]

Appendices are supporting material that has not been peer-reviewed.

A FINDING THE GOOD *IPD* PARAMETRIZATIONS

Since the *IPD* algorithm can be controlled by multiple parameters (see § 3), the question emerges on how to choose a suitable parametrization. Thus, we next aim to identify parameter sets that enable the *IPD* to run at a major tier-1 ISP. To do so, we conduct a systematic parameter study that uses a factorial design [26] to evaluate a total of 308 distinct *IPD* parameter configurations against traffic from a tier-1 ISP (see § 4). In the first step (§ A.1), we perform a factor screening of 108 configurations to identify parameter ranges that impact the algorithm. This step identifies parameters that have no impact and can thus be fixed and parameter sets for which the *IPD* fails and are thus to be avoided. In the second step (§ A.2), we study the impact of the working parameters to study *how* they impact *IPD* performance. For a fine granular evaluation, this step studies 200 configurations with more parameter levels than the first (see Table 2). We show that many parametrizations are practically feasible; while they do not largely impact the *IPD* accuracy, they mostly impact its resource consumption. This enables us to derive a parametrization that is used by the tier-1 ISP in its deployment.

Experimental setup. In this study, we evaluate the *IPD* with different parameter sets against the 25 hours of Netflow collected by the ISP *on premise*. Since the algorithm is deterministic, each parameter set was run once. We defined three metrics to evaluate the algorithm's performance:

- **Accuracy:** To measure accuracy, we compare the results of the *IPD* algorithm with Netflow data and determine how many flows were correctly classified by the *IPD*.
- **Stability Duration:** This metric measures how long the *IPD* algorithm classifies a stable ingress of a prefix on the same link, depending on factors such as split and join events. For an accurate predication of the ingress links, longer stable periods are preferred. For each set of parameters, we compute a CDF representing the stability periods of each range at an ingress point. To provide an objective measure of comparison, we evaluate these CDFs in relation to an ideal stability distribution, employing the Kolmogorov-Smirnov Distance as a metric [18]. Given the absence of prior research on the distribution of stable phases for prefixes at an ingress point, we explore various potential distributions, such as normal, lognormal, Weibull, and Pareto. Through these metrics, we gain the capability to accurately gauge the similarity between the observed stable periods and the ideal distribution.
- **Resource Consumption:** This metric measures the *IPD* algorithm runtime for a time bucket and the RAM usage. In a productive environment, operators can opt for a more resource-efficient configuration, with the trade-off being lower prefix specificity.

A.1 Finding Suitable Parameter Ranges

To determine the impact of *IPD* configuration parameters on the target metrics, we first perform a factor screening experiment to find out the algorithm's limits followed by a parameter study in a full factorial design to evaluate the performance of the algorithm. In light of this factorial design, we will refer to *IPD* configuration parameters such as q or $cidr_{max}$ as "factor" and their setting as "level". We implemented a conditional parameter setting for the factor levels of $n_{cidr}factor$ and $cidr_{max}$ for IPv4 and IPv6, respectively, to ensure that both factors are always set together, thereby avoiding any confounding effects that may arise if they were varied independently.

To identify the factors contributing to the performance outcomes of our metrics, we performed an Analysis of Variance (ANOVA) analysis for each metric [32]. ANOVA is a statistical method to test if differences in the mean of two groups are systematic or simply random. In our case, we use the ANOVA to check if parametrizations have a systematic impact on *IPD*, expressed in the different metrics.

Parametrizations to be set static or to avoid. Our factor screening experiment yields two important insights: (a) The factors $decay$ and e do not have a significant effect on the performance metrics, and thus, they can be fixed to a single value for subsequent experiments. (b) We find certain parameter sets for which the *IPD* algorithm fails. A $cidr_{max}$ value that is too low, such as 12 for IPv4, does not result in many classifications, while a value that is too high causes a significant increase in runtime and memory consumption. Furthermore, if the parameter q is less than or equal to 0.5, some ingress points may be classified ambiguously. For the parameter study experiment we have carefully chosen factor levels that ensure all parameter sets produce meaningful results, allowing us to investigate their influence on the presented metrics. Table 2 shows the factors and levels that we applied.

A.2 How do Paramerizations Impact *IPD*

We now study how the suitable parameter ranges identified in (§ A.1) impact the *IPD* performance.

Accuracy. We find that the parametrization does *not* influence the *IPD* accuracy. We find consistent accuracy figures for all parameter sets, with an average accuracy of 90.8 %. Later, we will show in § 5.1 that the *IPD* algorithm works accurately in deployment and our results in this parameter study are consistent with the later deployment.

Stability Duration. We find the parameters q and $cidr_{max}$ to influence the duration of stable phases (see Figure 19). Both, the

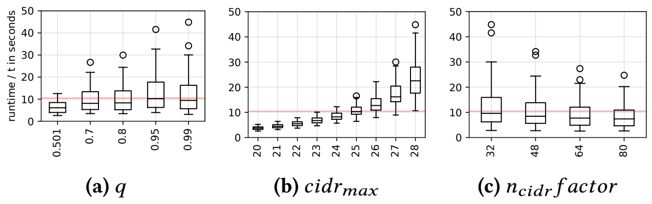


Figure 20: *IPD* runtime and resource consumption metric: Since higher $cidr_{max}$ result in larger classification tries (data structure), this parameter impacts the resource consumption exponentially.

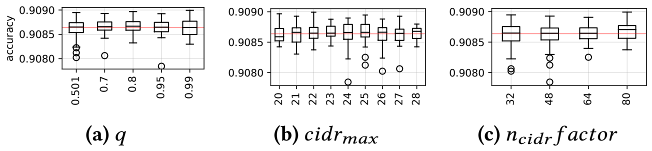


Figure 18: Effect plots showing the *IPD* accuracy metric for different parameter sets. The results show that the parametrization does not influence the *IPD* accuracy.

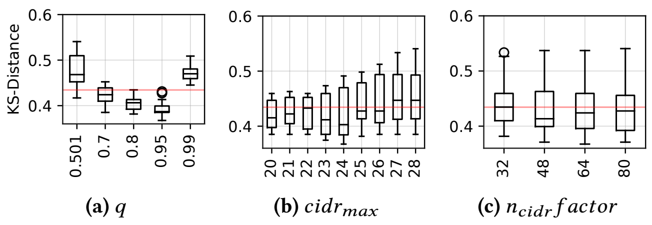


Figure 19: Effect plots of the prefix stability. Lower KS distance values indicate a higher similarity to the ideal distribution. The parameters $cidr_{max}$ and q impact the stability.

Kolmogorow-Smirnow distance and the average prefix stability reveal a similar trend: higher values of q lead to longer stable phases. The smallest distance to the ideal distribution is achieved when $cidr_{max}$ is set to 24. Based on the average parameter values, we observed that setting $cidr_{max}$ to more specifics than /25 results in longer stable phases.

Resource Consumption. The algorithm's runtime and memory usage are impacted by the number of ranges to be classified. Increasing $cidr_{max}$ results in finer-grained classification but also exponentially increases the number of ranges to be checked. Figure 20 illustrates this effect. Changes to q or $n_{cidr}factor$ also impact the metrics, albeit to a lesser extent. Configurations with high q values combined with low $n_{cidr}factor$ values lead to more frequent prefix dropouts. In such cases, only a few samples are necessary for classification, but the high value of q permits only a small amount of noise. As a result, situations can arise where the state of each (masked) IP must be held for each range until reclassified (as discussed in § 3). This in turn leads to longer runtimes and more RAM consumption.

Takeaway. *A broad range of possible parametrizations exist that will work in practice. While they have relatively little impact on accuracy, they mostly impact convenience settings such as resource consumption or mapping stability and can thus be adjusted to practical needs.*

Parametrization at the tier-1 ISP. In the deployment, we opted for a q value of 0.95, which results in longer stable phases on average. We use $n_{cidr}factor$ values of 64 and 24 for IPv4 and IPv6, respectively, as a good trade off between resource consumption and prefix stability. Since the ISP required to prefixes specific enough to capture ingress point changes due to CDNs and traffic engineering, we set $cidr_{max}$ to /28 and /48 for IPv4 and IPv6, respectively.

B EXAMPLE IPD OUTPUT TRACE

We show an example raw output of the *IPD* in Table 3. The confidence value $s_{ingress}$ is derived from the sum of all samples (column $s_{ipcount}$). The "ingress" column first shows the router with the highest current sample count for the range (the most prevalent ingress candidate). In parentheses, *all* ingress points and their traffic share are shown. This would enable congestion management, e.g., as in [22]. Note: For stage 2 of the algorithm, as it is currently used in deployment, this data is further filtered to include only prevalent ingress points.

Table 3: Raw *IPD* output. 6 years of *IPD* data from deployment at a tier-1 ISP serve as our main data set.

timestamp	ip	$s_{ingress}$	$s_{ipcount}$	n_{cidr}	range	ingress
1605571200	4	0.997	4812701	6144	x.y.0.0/16	C2-R2.4(C2-R2.4=4798963,C2-R3.54=12220, ...)
1605571200	4	1.000	1503386	543	x.y.104.0/23	C3-R20.7(C3-R20.7=1503296,C3-R20.14=90)
1605571200	4	1.000	4228708	543	x.y.106.0/23	C2-R30.1(C2-R30.1=4228502,C3-R20.24=201, ...)
1605571200	4	0.974	2262	192	x.y.65.64/26	C1-R1.1(C1-R1.1=2204,C3-R11.10=58)
1605571200	4	0.999	2441	192	x.y.149.192/26	C4-R11.21(C4-R11.21=2438,C3-R4.11=3)
1605571200	4	0.510	29996	96	x.y.65.32/28	C1-R1.1(C1-R1.1=15305,C3-R11.10=14691)
1605571200	4	0.722	433064	96	x.y.65.48/28	C3-R11.10(C3-R11.10=312631,C1-R1.1=120433)