AS-level BGP community usage classification

Krenc, Thomas; Beverly, Robert; Smaragdakis, Georgios

DOI
10.1145/3487552.3487865

Publication date
2021

Document Version
Final published version

Published in
IMC 2021 - Proceedings of the 2021 ACM Internet Measurement Conference

AS-Level BGP Community Usage Classification

Thomas Krenc
Naval Postgraduate School
tkrenc@nps.edu

Robert Beverly
Naval Postgraduate School
rbeverly@nps.edu

Georgios Smaragdakis
TU Delft
g.smaragdakis@tudelft.nl

ABSTRACT

BGP communities are a popular mechanism used by network operators for traffic engineering, blackholing, and to realize network policies and business strategies. In recent years, many research works have contributed to our understanding of how BGP communities are utilized, as well as how they can reveal secondary insights into real-world events such as outages and security attacks. However, one fundamental question remains unanswered: "Which ASes tag announcements with BGP communities and which remove communities in the announcements they receive?" A grounded understanding of where BGP communities are added or removed can help better model and predict BGP-based actions in the Internet and characterize the strategies of network operators.

In this paper we develop, validate, and share data from the first algorithm that can infer BGP community tagging and cleaning behavior at the AS-level. The algorithm is entirely passive and uses BGP update messages and snapshots, e.g. from public route collectors, as input. First, we quantify the correctness and accuracy of the algorithm in controlled experiments with simulated topologies. To validate in the wild, we announce prefixes with communities and confirm that more than 90% of the ASes that we classify behave as our algorithm predicts. Finally, we apply the algorithm to data from four sets of BGP collectors: RIPE, RouteViews, Isolario, and PCH. Tuned conservatively, our algorithm ascribes community tagging and cleaning behaviors to more than 13k ASes, the majority of which are large networks and providers. We make our algorithm and inferences available as a public resource to the BGP research community.

CCS CONCEPTS

• **Networks** → *Network protocols*; **Network management**.

KEYWORDS

Border Gateway Protocol (BGP), BGP communities.

ACM Reference Format:
Thomas Krenc, Robert Beverly, and Georgios Smaragdakis. 2021. AS-Level BGP Community Usage Classification. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3487552.3487865

1 INTRODUCTION

Border Gate Protocol (BGP) communities [16], an optional transitive attribute attached to routing announcements, are widely used to communicate information and actions within and between Autonomous Systems (ASes). The flexibility to define communities, tag routes with communities, and automatically take a prescribed action on aggregates of tagged routes, has enabled a variety of common uses including traffic engineering, remote triggered blackholing (RTBH) [4], and other custom network policies and business strategies [7].

Community values are simply byte strings, and there is no globally defined semantic to these values beyond a handful with special meaning. ASes define particular community values either for internal network use, e.g. to tag the geographic location where prefixes are received, or for signaling to peers, e.g. to request AS path prepending. While it is a general convention that the high-order bytes of the community are set to the AS Number (ASN) of the entity defining the community meaning, the low-order bytes are set according to each network's policy. And while some networks publicly publish their community conventions, many networks do not. Thus, for both operational and research purposes, it is often difficult to understand the intent of communities that are seen attached to routes, for instance at route collectors.

Even more fundamentally, the macro-level community usage behavior of different ASes is unknown. As BGP announcements propagate, individual ASes may arbitrarily ignore, add, modify, or remove communities. In this paper, we develop, validate, and employ a passive algorithm to infer per-AS community usage. At its core, our algorithm implements a system of constraints that groups ASes according to those that do or do not tag announcements with communities, namely, *tagger* and *silent* ASes, as well those that remove or ignore communities in updates, namely, *cleaner* and *forward* ASes. When our algorithm has conflicting information, for instance due to complex per-peer selective behavior, it labels the AS as *undecided*, and if it cannot make any inference, *none*.

We first characterize the coverage and accuracy of our algorithm, along with the sensitivity of its parameters, using simulations of large AS topologies. We then validate a subset of our inferences by using the PEERING testbed [23] to announce a prefix under our control into the BGP. Using data from three large route collectors, we show that our algorithm correctly infers the community usage behavior of 90% of the ASes we predict, for those ASes in observed paths.

Understanding per-AS community usage behavior is important to not only measurement research that utilizes communities, but also in uncovering important properties of operational networks in the wild. For instance, work on measuring community propagation and security [26], community-driven routing load and unnecessary updates [15], community-based outage detection [8] and blackholing [11], and hijack detection schemes [24] can all benefit from a grounded per-AS understanding of BGP community usage behavior. Thus we see our work filling an important gap for the measurement and network research community, akin to AS relationship inference [3, 9]. Our contributions include:

- A novel passive algorithm to infer per-AS BGP community usage behavior.
- Analysis of simulated and real-world experiments demonstrating the coverage, accuracy, and sensitivity of our algorithm.
- Application of our algorithm to data from four major route collectors.
- We make our algorithm and database of community usage behavior publicly available to the network measurement community [5].

The remainder of this paper is organized as follows: In Section 2 we introduce related work in this area and provide background information on BGP and BGP communities in Section 3. In Section 4 we introduce the data sets that we use throughout this work and the sanitation steps involved. In Section 5, we present our mental model and derive constraints to implement in to the inference algorithm. In Section 6 we evaluate its performance using generated ground truth data sets. In Section 7 we present our classification results using publicly available routing information. We conclude in Section 8.

2 RELATED WORK

As the Internet's inter-domain routing glue, the BGP [19] is critical protocol in the operation of the network. It is therefore natural that the behavior, evolution, stability, and security of BGP have been extensively analyzed, e.g. [6, 12, 17, 27] to name a few studies. Similarly, the adoption of BGP communities [16] has driven research into the ways in which communities are used, as well as how they can reveal important properties and events in the Internet.

Benoit *et al.* provided the first taxonomy of BGP communities [7]. Since then, community usage has continued to increase. Specifically, among more than 78,000 unique regular communities observed in the recent routing data we analyze, there are more than 6,300 distinct high-order two-byte values – indicative of the wide variety of use and breadth of adoption. Giostas *et al.* provided one of the first examples of using communities as a measurement tool, by leveraging observed activity of blackhole communities to estimate the prevalence, frequency, and duration of denial-of-service attacks [11]. Subsequent work used communities as a proxy to detect peering infrastructure outages [8].

Further motivating our work, Streibelt *et al.* explored the potential attack vectors using BGP communities, and their feasibility due to a general lack of community filtering [26]. Streibelt's work showed that the lack of common understanding in community values, combined with their transitive property, leads to communities propagating further than intended. As a result, malicious actors in the BGP system can potentially blackhole or re-route traffic.

Moreover, while Streibelt *et al.* explore the extent to which communities propagate, they do not attempt to determine whether individual ASes are taggers (or silent). Similarly, Streibelt examines filtering and propagation of communities on a per-edge basis in the AS graph, but do not reconcile conflicting information or make per-AS filtering classifications. Our algorithm holistically takes into account filtering and tagging behaviors and is specifically tuned to make precise inferences. Last, whereas prior work was entirely inferential, we use lab experiments, simulations, and live announcements to validate and understand the limits of our approach.

Also closely related and motivating our investigation, Krenc *et al.* explore the BGP update message load and impact due to communities [15] and permissive propagation. In particular, Krenc's longitudinal study finds a large number of unnecessary updates that could be avoided, benefiting security and performance, with stricter community filtering by providers.

Our algorithm could thus usefully be combined with these prior efforts to provide a richer understanding of both which ASes are filtering too permissively, as well as which ASes are vulnerable to attack or contributing to unnecessary routing traffic.

3 BACKGROUND

The Internet consists of more than 70K Autonomous Systems (AS). An AS comprises one to multiple networks under the same administrative control and it is identified by the AS Number (ASN), a 16-bit integer value. As with many resource identifiers in the Internet, the ASN is a scarce resource which limits the number of public ASes in the Internet to 2^{16}. Today, while not all 16-bit ASNs are actively routed, the 16-bit address space is allocated. In order to allow more ASes to participate in the Internet, the ASN address space has been expanded to 32 bits [28].

ASes exchange traffic with each other to allow communication among all Internet users. Thereby, the traffic flow is determined by contractual business relationships between pairs of ASes. In the following, we explain how ASes exchange routing information via the Border Gateway Protocol (BGP) and introduce the BGP community attribute, which allows ASes to signal information across the AS level graph.

3.1 Border Gateway Protocol

Business relationships are implemented using the Border Gateway Protocol, often based on the size and the traffic volumes of the involved ASes. There exist two primary types of business relationships: peer-to-peer and provider-customer. ASes typically form peer-to-peer relationships when both networks have a similar volume of traffic to exchange. On the other hand, in a customer-provider relationship, one of the ASes is the customer who pays the provider to exchange traffic. This is typically the case, e.g., when a small network requests transit to the global Internet via a larger network.

Geographically close ASes can interconnect directly at convenient locations and thus potentially circumvent transit costs and keep traffic local. Popular peering locations are Internet Exchange Points (IXP) which provide a shared infrastructure, large hubs in metropolitan areas, for hundreds of ASes to exchange traffic directly, either publicly or privately. In order to facilitate multi-lateral peering, IXPs offer router servers as value added service [20]. Also, IXPs offer remote peerings where no physical presence by the AS is required [10]. Given the multitude of possibilities two ASes can interconnect, it is little surprising that they can have different business relationships at different locations which further increases the complexity of relationships [9].

In order for an AS to become globally reachable, it sends prefix announcements – encapsulated in BGP update messages – to neighboring ASes, which in turn forward the prefix announcements to their neighbors, and so on. The path a prefix is forwarded along is called the AS path. An AS path p is a sequence of n ASNs: $A_1, A_2, \ldots, A_n$, while each ASN corresponds to the AS that has forwarded the announcement.

For purposes of exposition in this paper, for any AS path p and AS $A_x \in p$, we distinguish between *upstream* ASes of A_x, i.e., all ASes $A_i \in p, i < x$, and *downstream* ASes, i.e., all ASes $A_j \in p, j > x$. We refer to A_1 as the collector *peer*, or simply peer AS,

and A_n as the *origin* AS, while the announcement originates at A_n and is forwarded *upstream* towards A_1.

Our inference algorithm relies on views of BGP routes, in particular AS paths and community attributes, which we obtain via public route collector "looking glasses," (Section 4). Peer ASes forward announcements to route collectors, where the announcements are archived and made publicly accessible to, e.g., network operators to monitor or debug their routing configurations.

We further distinguish between *leaf* and *transit* ASes: Given a set of AS paths P, AS A_x is a *leaf* AS, $if\ \nexists p \in P$ with $A_x \in p \wedge x \neq |p|$. Conversely, A_x is a *transit* AS, $if\ \exists p \in P$ with $A_x \in p \wedge x < |p|$. In other words, a leaf AS only originates prefixes but never forwards other prefixes, because it has no *downstream* neighbors. *transit* ASes on the other hand forward announcements from *downstream* ASes.

3.2 BGP communities

The BGP Communities Attribute [16] is appended to updates and is used to aggregate routes with common properties. It is a variable length attribute and can store multiple communities. Also, the community attribute is a transitive attribute which allows communities to be propagated across multiple ASes.

A regular BGP community is simply a 32-bit integer that is denoted in the form α:β, where typically α represents the AS that defines the value β. Thus, every AS can define its own values without collisions. Note that this convention applies only to encoding 16-bit ASNs. In order to accommodate 32-bit ASes as well, the BGP Large Communities Attribute has been introduced [13]. A large community is a 3x32-bit integer denoted in the form α:β:γ, while α represents a 32-bit ASN and β and γ are additional 64 bits for the community value. For simplicity, we refer to α in both community variants as the *upper* field in the remainder of this work[1]. In order to determine the community usage of 32-bit ASes, in this work we consider large communities as well. Note that 16-bit ASes can also use large communities.

Source of communities: Communities can be used to simply convey meta information, e.g., the location where an announcement has been received. In the following example, AS Z propagates information to the *upstream* AS X:

$$X \xleftarrow{Z:*} Z$$

[1]RFC8092 calls the field the *Global Administrator*.

Z:$*$ is also referred to as *informational* community [25]. However, communities are not always set by the AS that defines it. For example, communities can instruct other ASes to perform a certain task, e.g., blackholing or path prepending. In the following example, AS Z instructs AS X to perform a specific action on the announced routes, indicated by the value defined by AS X:

$$X \xleftarrow{X:*} Z$$

Here, X:$*$ is also referred to as *action* community.

We note that the source of communities can be ambiguous: Some ASes define communities (action and informational) using a non-public ASN instead of their own in the *upper* field, e.g., ASes with a 32-bit ASN using regular (32-bit) communities, or they use standardized, well-defined communities [2, 16]. Further, while route servers at IXPs utilize communities as well, their ASN typically does not appear in the AS path which further obfuscates the source those communities.

Because the community attribute is an optional attribute, and BGP provides no mechanisms to validate the source, any AS along the AS path may add, modify or delete communities. Thus, it is not guaranteed that the *upper* field corresponds to the tagging AS.

Our visibility into the BGP system is limited by available collectors and the ASes to which they peer. Since we cannot reliably determine the source of a community, we at a minimum require its *upper* field to be present in the associated AS path. For that purpose, we group communities based on the *upper* field in relation to the position of the *upper* field in the AS path. We define the following community source groups:

- A *peer* community is a community where the *upper* field corresponds to the peer ASN in the AS path, i.e., A_1.
- In a *foreign* community, the *upper* field does not equal the peer's ASN, but any other ASN in the AS path, i.e., A_i, i>1.
- *stray* is a community with the *upper* field representing a public ASN, but the ASN is not in the AS path.
- *private* is a community with the *upper* field representing a non-public ASN, i.e., private, reserved, not assigned or allocated, etc.

Note, a *peer* community in given AS path p_1 can appear as a *foreign* community in AS path p_2.

Our inference algorithm, which we introduce in Section 5, necessarily ignores *stray* and *private* communities, since there is no indication as to which AS set those communities without additional knowledge beyond what is available in a BGP announcement. For *peer* and *foreign* communities, we assume the community was set by the AS that corresponds to the upper 2 bytes (or upper 4 bytes) of the community. We consider all community types when we verify the correct functioning of our inference method in Section 7.2.

3.3 Community usage

We define community usage to describe the way ASes generally configure their routers to tag routes with communities or filter them, irrespective of the semantic meaning of community values.

3.3.1 Mental Model. In order to better understand community usage in the wild, we distinguish ASes according to their individual community usage. To start with, we ask for two basic properties: (1) whether or not an AS adds its own communities to the community attribute of an announcement, and (2) whether or not it cleans existing communities received in the announcement (set by other ASes along the AS path). We refer to the first property as *tagging* behavior and the second property as *forwarding* behavior.

Since any AS along the AS path can arbitrarily modify the community attribute, we narrow down the two properties to reflect consistent behavior, as described in Section 3.2. For the *tagging* behavior, we define a *tagger* AS that defines and sets informational communities internally in a consistent and automated fashion and forwards them on external BGP sessions. The counterpart, a *silent* AS, does not set its own communities or does not forward them on external sessions. In addition, we specify the *forwarding* behavior to reflect the forwarding of communities set by *tagger*s. Thereby, a *forward* AS forwards communities on external sessions set by other *tagger*s. Conversely, a *cleaner* is an AS that does not forward communities on external sessions. In summary:

- *tagging* behavior
  - *tagger*: A *tagger* AS adds its own communities in a consistent and automated fashion, i.e., informational communities, and forwards them on external links.
  - *silent*: A *silent* AS may use its own communities internally, but it does not forward them on external links.
- *forwarding* behavior
  - *forward*: A *forward* AS does not remove communities added by other *tagger* ASes and forwards them on external links.
  - *cleaner*: A *cleaner* AS removes communities set by other *tagger* ASes, either upon receiving or upon forwarding on external links.

Every AS has a *tagging* and *forwarding* behavior, e.g. an AS can be *tagger* and *forward* at the same time (but not *tagger* and *silent* at the same time). Note, that our definitions do not exclude the possibility that communities are used only AS-internally and filtered upon reannouncing. However, this case is difficult to classify and requires AS-specific insights that go beyond the scope of this work.

3.3.2 Formal Model. To capture the *tagging* behavior of AS A we define a function that returns a non-empty community set with communities $A{:}*$ if A is a *tagger* and an empty community set if A is a *silent* AS:

$$tagging(A) = \begin{cases} A{:}*, & if\ is\_tagger(A) \\ \emptyset, & if\ is\_silent(A) \end{cases}$$

Analogously to the *tagging* function, we define a *forwarding* function that captures the *forwarding* behavior of AS A:

$$forwarding(A, input) = \begin{cases} input, & if\ is\_forward(A) \\ \emptyset, & if\ is\_cleaner(A) \end{cases}$$

Here, the function returns the community set *input* from a neighboring AS if A is a *forward*, and an empty community set if A is a *cleaner* AS. Thus, the combined community set *output* of AS A is simply the union of the *tagging* and *forwarding* functions:

$output(A) = tagging(A) \cup forwarding(A, input(A))$

When, for example, the *tagging* and *forwarding* functions for AS A return an empty set (because A is a *silent* and *cleaner* AS) then the community set output $output(A)$ will be empty as well.

In Section 5, we discuss implications of our mental model and introduce an inference method that classifies the community usage behavior of an AS A, based on its community set output $output(A)$.

3.3.3 Selective tagging. Thus far, we assume that ASes employ a uniform community usage policy for all BGP peers. The simplest form of community configuration is to enable or disable *tagging* and *forwarding* behaviors identically for all neighbors. Naturally, more complex configurations exist. A large AS with sessions to multiple ASes may perform different actions for different neighbors, e.g. based on the business relationship. For instance, some ASes configure their BGP to only propagate communities (e.g. geolocation communities) to customers or to BGP collectors, but not to peers or providers. Depending on the visibility afforded by different route collectors, our algorithm can discover such selective community usage behavior. However, as our primary goal is to definitively classify those ASes with consistent behavior with high precision, our algorithm labels these instances as *undecided*.

In a worst case scenario, a peer AS is configured to not forward any communities to the collectors but to customers or peers. Our algorithm can potentially misclassify this behavior as *silent*, which can have a detrimental impact on the inferences of other ASes, as we will discuss in Section 5.4. We explore the performance of our algorithm in the presence of consistent and selective community behaviors in Section 6.

4 DATA SETS

Our inferences and analyses utilize AS path / community set pairs, which we denote as the tuple $(path, comm)$. We obtain these tuples from public BGP collectors[2].

Archived BGP update messages contain only a subset of all ASes and AS paths. There exist a significant number of ASes that do not appear in update data because they do not re-announce their routes frequently. These stable ASes and paths can only be found in RIB snapshots, which are also available from route collectors. Similarly, updates capture the intermediate changes to the routing system which are not available from RIBs. In this work we consider the combined use of archived updates and RIB snapshots.

4.1 Download and sanitation

We download RIBs as well as updates encoded in the Multi-Threaded Routing Toolkit (MRT) format from three major route collector projects: RIPE [21], RouteViews [22], and Isolario [14] for the complete day May 19, 2021. While we focus mostly on this date, we also download and analyze other data sets; each time at day granularity. We also obtain PCH [18] updates, however, PCH does not provide RIBs that include the community attribute. To avoid impairing our inferences due to missing stable ASes from RIBs, we exclude the PCH data set from most of our analyses.

Before analyzing the routing data, we first filter and transform it so as not to impart unintentional bias into our results. We remove routing information that includes unallocated prefixes or ASNs

[2]BGP collectors used: **RIPE**: rrc00-26, **RouteViews**: route-views{2,3,4,6}, amsix, chicago, chile, eqix, flix, gorex, isc, kixp, jinx, linx, napafrica, nwax, phoix, telxatl, wide, sydney, saopaulo, sg, perth, peru, sfmix, siex, soxrs, mwix, rio, fortaleza, gixa, bdix, bknix, uaeix, **Isolario**: Alderaan, Dagobah, Korriban, Naboo, Taris, **PCH**: 236 collectors.

Input data	RIPE	RouteViews	Isolario	d_{May21}	PCH
Entries total	2,242M	4,679M	2,088M	9,010M	532M
incl. RIB entries	1,022M	3,160M	1,275M	5,458M	0
Uniq. $(path, comm)$	46M	30M	16M	77M	1M
AS numbers	79,192	80,097	78,454	80,651	77,082
After cleaning	72,737	72,837	72,625	72,951	67,541
incl. Leaf ASes	60,418	60,838	60,834	60,420	56,703
incl. 32-bit ASes	31,147	31,193	31,082	31,239	29,077
Collector peers	525	291	108	766	1,304
Communities	12,133M	16,513M	11,056M	39,703M	2,185M
incl. large	2,614M	2,196M	2,283M	7,093M	660M
Unique communities	75,681	73,720	67,816	84,186	35,804
incl. large	4,147	4,605	3,581	5,326	3,383
Uniq. *upper* field (regular)	5,904	6,091	5,713	6,385	3,303
Uniq. *upper* field (large)	378	357	358	384	884
Uniq. *upper* field (both)	6,160	6,340	5,958	6,643	4,292
w/o *private*	5,660	5,739	5,474	6,025	3,931
w/o *stray*	4,316	4,373	4,189	4,579	2,574

Table 1: Data sets overview: RIBs + updates for collector projects RIPE, RouteViews, Isolario, PCH in May 19, 2021. d_{May21} represent the aggregation of RIPE, RouteViews, and Isolario. PCH is treated separately due to missing communities in RIB snapshots.

using current allocation information from the regional registries. We further remove *AS_SET*s from AS paths which usually occur when routes are aggregated. Also, we prepend the *Peer AS Number* from every MRT message to the AS path, if the first ASN (A_1) does not equal the *Peer AS Number* [1]. This is the case for route servers at IXPs which typically do not participate in the routing decision process, but still can modify the community attribute. Finally, we remove path prepending from AS paths by collapsing identical ASNs in succession.

In Section 7 we provide inference results for each individual collector project. However, for the majority of our analyses, we consider the aggregated data set of RIPE, RouteViews, and Isolario. In the remainder of this work, we refer to this data set as d_{May21}. In Section 5 we use the AS paths from the aggregated data set d_{May21} in order to generate ground truth data sets to validate the algorithm.

4.2 Data overview

Table 1 provides an overview of the available data sets. Each of the three data sets – RIPE, RouteViews, and Isolario– provide us with Billions of entries with around 45% obtained from RIBs in the case of RIPE, 67% RouteViews, and 61% Isolario. Note that the route collector projects employ different update file binnings and snapshot intervals. There are more than 9B entries in the aggregate d_{May21}, of which we extract 77M unique $(path, comm)$ pairs.

Overall, the number of distinct ASes observed in RIPE, RouteViews, and Isolario are comparable: we find approximately 72K ASes which consist of ~60K leaf ASes and ~12K transit ASes. Furthermore, there exist around 31K 32-bit ASNs in the data set. An AS can be a leaf AS and 32-bit AS at the same time. Regarding the number of collector peers, PCH has the highest number followed by RIPE, RouteViews, and Isolario. Note that a peer AS can peer at multiple BGP collectors.

Lastly, looking at the communities, we observe ~40B communities in d_{May21}, of which around 7B stem from the large community attribute. Overall, there are 84K unique communities. Those communities exhibit 6,643 unique *upper* fields in both community variations, while regular communities contribute 6,385 unique *upper* fields and large communities 384. Recall from Section 3.2 that 16-bit ASNs can also be encoded in large communities. Interestingly, PCH shows the highest number of unique *upper* fields in large communities.

As mentioned in Section 3.2, our inference method has no use for *private* and *stray* communities. When we discard communities where the *upper* field resemble a *private* ASN, 6,025 remain. When we further discard communities where the *upper* field is never in the corresponding AS path (i.e. *stray*), we end up with 4,579 unique *upper* fields. Those are candidates for *tagger* ASes.

5 INFERENCE METHOD

While some ASes publicly document their community definitions, there is no documentation as to how those ASes generally set communities on external sessions, e.g., consistent or selective *tagging*, and whether or not they filter communities set by other *tagger*s. Thus, the community usage has to be inferred. The purpose of the inference method is to infer the BGP community usage per AS by observing the community set output.

Recall from Section 3.3 that it is possible to utilize the community set output $output(A)$ of an AS A to make statements about its community usage. Ideally, the more ASes that propagate their community set output to BGP collectors the better the visibility on the community usage of non-peer ASes. Unfortunately, only a small fraction of ASes (~1,751 in $d_{May21} \cup$ PCH) peer with BGP collectors. The determination of community usage behavior of the remaining ASes is thus not trivial. Therefore, we devise an inference method that infers community usage only by utilizing the community set output of BGP collector peers.

In Section 5.1 we state the implications of our mental model on the inference algorithm, which we detail in Sections 5.2 through 5.6. Also, we provide a discussion of our column-based approach in Section 5.7. In Section 6 we verify the functioning of the algorithm by generating and using customized ground truth data sets with known community usage behavior.

5.1 Implications from mental model

As mentioned above, our inference method is limited to the community set output of collector peers, i.e., given a collector C, an AS path p, and the community set *output*(A_1) of the peer AS, such that: $C, A_1, A_2, \ldots, A_n | output(A_1)$. The *tagging* function of any peer AS A_1 can be directly observed in its community set output, because *output*(A_1) = *tagging*$(A_1) \cup$ *forwarding*$(A_1,$ *input*$(A_1))$. Consider the following example where ASes X and Y peer with collector C:

$$C \xleftarrow{X{:}*} X$$
$$C \xleftarrow{\emptyset} Y$$

Assuming that every AS uses its own ASN in the *upper* field of the community, we can safely conclude that in this context AS X is a *tagger* and Y is a *silent* AS.

Further, it is possible to use *output*(A_1) to infer the community usage of any AS $A_x \in p$. Note that *input*(A_x) is the community set *output* of *downstream* neighbor A_{x+1}, i.e.: *input*(A_x) = *output*(A_{x+1}). Since *output*() is recursive, *output*(A_1) can contain information about the community usage of *downstream* ASes. However, the information can be ambiguous or hidden, as we will explore next.

5.1.1 Noise. The simplest form of ambiguity stems from the fact that any AS along the AS path can arbitrarily modify the communities in the community attribute. The resulting *noise* can lead to a wrong perception of community usage. Consider the following example:

$$C \xleftarrow{Y{:}*} X \xleftarrow{?} Y \xleftarrow{?} Z$$

Without additional information beyond what exists in BGP data we are not able to determine the source of community $Y{:}*$. There are multiple interpretations: (a) AS Y might have tagged the route with an informational community to indicate the ingress location. (b) AS Z might have used an action community defined by AS Y to instruct Y to perform a specific action on that route. (c) AS X could have unconventionally defined and set this community.

If AS Y is a visible *tagger* AS (not hidden behind a *cleaner* AS), then (b) and (c) will not impact our classification. If, however, Y is a *silent* AS, (b) and (c) can lead to a misclassification of AS Y to be a *tagger*. In Section 6.4 we show that noise, e.g. caused by (b) and (c), has no significant impact on the inference of *tagger* and *forward* behaviors, but on the inference of *silent* and *cleaner* behavior. To minimize misclassifications, we allow an AS to be labeled *undecided*, if it cannot be clearly assigned *silent* or *tagger*.

5.1.2 Hidden behavior. Other forms of ambiguity emerge when the *tagging* and *forwarding* behavior is hidden behind a *cleaner* AS, or when there are no *downstream tagger* ASes to illuminate the *forwarding* behavior of upstream ASes. Let us look at an example where AS X has a *downstream tagger* Z:

$$C \xleftarrow{Z{:}*} X \xleftarrow{Z{:}*} Z$$

If we see the community $Z{:}*$ at collector C, we can conclude that AS Z is a *tagger* while AS X is *silent*. More importantly, the fact that we can observe $Z{:}*$ allows us to conclude that X is a *forward* AS (and not a *cleaner* AS). But it is not always as trivial. Consider now the following example:

$$C \xleftarrow{\emptyset} X \xleftarrow{?} Z$$

Not knowing a priori that Z is a *tagger* (C receives an empty community set), it is impossible for us to tell, whether (a) AS Z is *silent* and X is a *forward* AS, or whether (b) AS Z actually adds communities which are then removed by *cleaner* X.

These kind of ambiguities exacerbate the identification of consistent behavior as we show next. Consider the simplified sequence of updates at collector C, where X is a *forward* and Y is a *cleaner* AS:

$$C \xleftarrow{Z{:}*} X \xleftarrow{?} Z$$
$$C \xleftarrow{\emptyset} Y \xleftarrow{?} Z$$
$$C \xleftarrow{\emptyset} Y \xleftarrow{?} Z$$

From the collectors point of view, one could conclude that Z is a *tagger* because we observe and count the community $Z{:}*$. One could also count the two occurrences where Z appears silent ($\emptyset$). Given a configurable threshold, Z can then be classified either *tagger* or *silent*. However, this approach comes with two problems. First, finding the optimal threshold to work in the wild requires ground-truth which is not available. Second, the approach does not consider the possibility that Y is a *cleaner* AS, erroneously counting those instances as *silent*.

Since our goal is to identify consistent behavior with high precision, it is not sufficient to simply count the occurrences of communities (or the absence thereof). In addition to the existence of communities, our approach requires X and Y to be a *forward* AS in order to count the behavior of AS Z. Assuming that X is a *forward* AS and Y is a *cleaner* in the above example, we consider only the first update for counting, leading to the classification of Z as *tagger*.

5.1.3 AS-level periphery. Another form of hidden behavior shown by leaf ASes (as defined in Section 3.1). Leaf ASes have no *downstream* ASes and thus no *input*() to determine their *forwarding* behavior. We consider leaf ASes useful since their *tagging* behavior can still help to illuminate the *forwarding* behavior of *upstream* ASes.

5.2 Conditions

In order to deal with ambiguous or hidden information, and subsequently minimize wrong inferences, in the following we define conditions that fit our mental model and are required when counting the community usage per AS, see Section 5.3. If the conditions are not true, we cannot make any statement about the community usage of an AS A_x, given a $(path, comm)$ pair $C, A_1, A_2, \ldots, A_n | output(A_1)$.

Cond1: *is_forward*$(A_i) | \forall A_i, i{<}x$

In order to make any statement about AS $A_x \in path$ we need to make sure that all *upstream* ASes A_i, with $i{<}x$ are *forward*. Conversely, when a single *upstream* AS is a *cleaner*, e.g., if A_{x-1} is a *cleaner* AS, then $output(A_x)$ is hidden. Thus, $Cond_1$ must be fulfilled in order to determine the *forwarding* and *tagging* behavior of AS A_x ($forwarding(A_x, input)$ and $tagging(A_x)$).

Cond2: $is\_forward(A_j) \land is\_tagger(A_t) | \forall A_j \exists A_t, x<j<t$

In order to determine the *forwarding* behavior of AS A_x, an additional condition is required. Recall from Section 5.1.1 that in order to determine the *forwarding* behavior of an AS, it requires input from a *downstream tagger*, here A_t. Thus, if there are no *downstream tagger*s, or the *downstream tagger*s are hidden by a cleaner, we cannot make any statement about the *forwarding* behavior of A_x. For example, assume $is\_silent(A_{x+1}) \land is\_cleaner(A_{x+1})$ is true, then $output(A_{x+1}) = \emptyset$ and $forwarding(A_x, \emptyset) = \emptyset$.

5.2.1 Race conditions. Under certain topological conditions, both the *tagging* and *forwarding* behavior of ASes cannot be determined. Consider the following scenario: AS X is the only *upstream* provider of AS Y, while Y is the only *downstream* AS of X.

$$C \xleftarrow{?} X \xleftarrow{?} Y$$

In order to infer the *forwarding* behavior of AS X, at least one *downstream* AS is required, here AS Y, to be a visible *tagger* ($Cond_2$). However, to count the *tagging* behavior of AS Y in the first place, it requires all *upstream* ASes, here AS X, to be *forward* ($Cond_1$). Thus, AS X requires pre-existing knowledge about AS Y and vice versa which our inference method cannot resolve.

5.2.2 Summary. The community set $output(A_1)$ allows us to determine the *tagging* and *forwarding* behavior of ASes $A_i \in$ path p. However, noise, as well as *silent* and *cleaner* ASes that hide information about other ASes, can lead to ambiguities. In order to minimize wrong inferences, the *tagging* function requires $Cond_1$, and the *forwarding* function requires both, $Cond_1$ and $Cond_2$ to be true. However, race conditions can occur which leaves some ASes without class, i.e., *none*. In Section 6.4 we measure the impact of race conditions on our inference coverage.

5.3 Counting community usage

In order to infer the community usage per AS, we observe and count pattern that satisfy requirements derived from the implications in Section 5.1. Given input $C, A_1, A_2, \ldots, A_n | output(A_1)$, we increase the class counter for a given AS A_x if the following conditions hold true:

- *tagging*
  - $t[A_x]$ ++ $if\ Cond_1 \land A_x{:}* \in output(A_1)$
  - $s[A_x]$ ++ $if\ Cond_1 \land A_x{:}* \notin output(A_1)$
- *forwarding*
  - $f[A_x]$ ++ $if\ Cond_1 \land Cond_2 \land A_t{:}* \in output(A_1)$
  - $c[A_x]$ ++ $if\ Cond_1 \land Cond_2 \land A_t{:}* \notin output(A_1)$

Note that $Cond_1$ drops out if x=1, i.e., it is trivial to determine the *tagging* function of peer ASes. Also, if conditions for *forwarding* counters $f[]$ and $c[]$ are met, the conditions for *tagging* counters are met as well. If $Cond_1$ or $Cond_2$ are not met, none of the counters is increased. Based on those counters, we can query the *tagging* and *forwarding* behavior of A:

- *tagging*
  - $is\_tagger(A) = \frac{t[A]}{t[A]+s[A]} \geq tagger\_thrsh$
  - $is\_silent(A) = \frac{s[A]}{t[A]+s[A]} \geq silent\_thrsh$
- *forwarding*
  - $is\_forward(A) = \frac{f[A]}{f[A]+c[A]} \geq forward\_thrsh$
  - $is\_cleaner(A) = \frac{c[A]}{f[A]+c[A]} \geq cleaner\_thrsh$

For example, an AS A is a *tagger* if the share of $t[A]$ exceeds the *tagger* threshold. Recall from Section 3.3 that a *tagger* is an AS that sets its own communities in a consistent manner. Thus, we want the threshold to be as high as possible, but at the same time allow for exceptions, e.g., 99%. We explore different thresholds in Section 6.

5.4 Selective behavior

Not all ASes have a uniform *tagging* and *forwarding* behavior. An AS may add own and remove other communities selectively, e.g., on a per-session basis or based on the business relationship. Selective *tagging* can lead to a contradicting perception of community usage, i.e., the *tagger* counter $t[Z]$ for a given AS Z is increased in one instance, the *silent* counter $s[Z]$ for the same AS may be increased in a another. Consider the following two updates, where X and Y are *forward* ASes:

$$C \xleftarrow{Z:*} X \xleftarrow{Z:*} Z$$
$$C \xleftarrow{\emptyset} Y \xleftarrow{\emptyset} Z$$

We assume that AS Z is a *selective tagger* which adds communities only to customers, here AS X, but not to peers (AS Y). Based on that observation we increase the *tagger* and *silent* counters for the same AS Z. When both *tagging* counters $t[Z]$ and $s[Z]$ are such that neither $is\_tagger(Z)$ nor $is\_silent(Z)$ is true, we simply classify Z as *undecided*, see Section 5.5.

While *undecided* ASes invalidate $Cond_1$ and $Cond_2$ and thus exacerbate further inferences, misclassifications due to selective tagging can lead to further misclassification, as we show next. Consider the following scenario, where AS Z always tags routes toward the collector using its own communities, but never to any other AS:

$$C \xleftarrow{Z:*} Z$$
$$C \xleftarrow{\emptyset} X \xleftarrow{\emptyset} Z$$

If there are sufficient occurrences of the first update in the input data, AS Z will be classified as *tagger*. Now, assume that the algorithm attempts to determine the *forwarding* behavior of X using the second update. If there are sufficient occurrences visible, the algorithm will classify X as *cleaner*, because $is\_tagger(Z)$ is true, and the community $Z{:}*$ is not present.

5.4.1 Conclusion. Selective community usage is the root cause that lead to a limited view of community usage behavior. They can invalidate conditions $Cond_1$ and $Cond_2$ and thus exacerbate the counting. Subsequently, the missing counts of one AS can have a detrimental impact on the counting of another AS, etc. In Section 6.4 we show how our algorithm deals with selective *tagging*.

5.5 Classification

After the counting is done, we determine the community usage behavior, i.e., the *tagging* and *forwarding* behavior of AS A like this:

$get\_class(A) = get\_tagging(A) || get\_forwarding(A)$

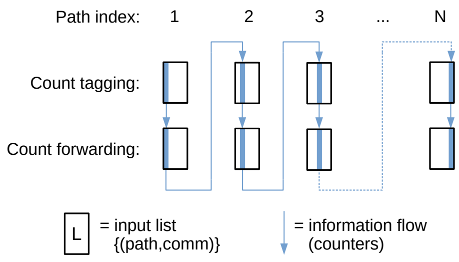


Figure 1: Inference algorithm workflow. Two passes over input $(path, comm)$ tuples: 1) to count *tagging* behavior and 2) to count *forwarding* behavior for each AS. Tuples are iterated per column (AS path index), starting with left-most ASes (A_1). Knowledge from one iteration is transferred to the next.

$$get\_tagging(A) = \begin{cases} n, & if\ t[A] + s[A] == 0 \\ else \begin{cases} t, & if\ is\_tagger(A) \\ s, & elif\ is\_silent(A) \\ u, & else \end{cases} \end{cases}$$

$$get\_forwarding(A) = \begin{cases} n, & if\ f[A] + c[A] == 0 \\ else \begin{cases} f, & if\ is\_forward(A) \\ c, & elif\ is\_cleaner(A) \\ u, & else \end{cases} \end{cases}$$

The function $get\_class(A)$ returns a two-character string which indicates the inferred *tagging* (first character) and *forwarding* (second character) behavior of AS A.

Both indicators can assume n (none) if the respective counters are zero. Recall, when any of the required conditions $Cond_1$ or $Cond_2$ cannot be fulfilled, none of the counters are increased. Thus, it can occur that all counters for an AS result in zero. When both *tagging* counters $t[A]$ and $s[A]$ for a given AS A are zero, there is not enough information to make any decision about the *tagging* behavior of A. The same applies to *forwarding* counters.

When *tagging* counters are available, the resulting *tagging* indicator can be t (*tagger*), s (*silent*), or u (*undecided*). Similarly, the *forwarding* indicator can be either f (*forward*), c (*cleaner*), or u. Note, that the resulting class depends on the thresholds that are used during counting, see Section 5.3.

The resulting two-character string for AS A can range among *tf* (*tagger-forward*), *sc* (*silent-cleaner*), or *nu* (*none-undecided*).

5.6 Algorithm

Considering the requirements for counting community usage (see Section 5.3) we note that counting each class depends on pre-existing knowledge of other ASes in the AS path, i.e., $Cond_1$ and $Cond_2$. In particular counters $c[]$ and $f[]$ require for conditions to hold true for *upstream* ASes as well as *downstream* ASes. Fortunately, it is trivial to count the *tagging* behavior of peer ASes (A_1), which can be used as initial knowledge to feed follow-up inferences. In order to meet the conditions $Cond_1$ and $Cond_2$ as often as possible, we design an algorithm that generates pre-existing knowledge by iterating over the input data a total of two times, in a column-based fashion (as opposed to row-based), see Figure 1. Iterating the data by column, i.e., the path index, we can utilize the recursive output of A_1 and apply the conditions as the algorithm progresses to higher indices. To show a sample implementation of the algorithm, we provide the corresponding pseudo code in Listing 1 in the appendix.

In the following, we describe the steps of the algorithm in detail. Given as input L = {(path,comm)}, a list of AS path and community set tuples, where $path = A_1, A_2, \ldots, A_n$ and $comm = output(A_1)$, we begin with counting the *tagging* behavior of peer ASes:

- path index 1
  - **count *tagging*** We begin by counting the *tagging* behavior at path index 1, i.e., A_1 of every AS path. For that, we increase the *tagger* counter $t[A_1]\ if\ A_1{:}* \in output(A_1)$. Otherwise, we increase the *silent* counter $s[A_1]$. Note that $Cond_1$ drops out at path index 1, since there are no upstream ASes.
  - **count *forwarding*** Next, we use the previously obtained *tagging* counters to fulfill $Cond_2$ of *forwarding* counters $i[]$ and $c[]$, and count the *forwarding* behavior at path index 1. We increase the *forward* counter $i[A_1]\ if\ A_t{:}* \in output(A_1)$. Else, we increase the *cleaner* counter $c[A_1]$.
- path index $i > 1$ ($Cond_1$ kicks in)
  - **count *tagging*** We use *forwarding* counters from previous steps to fulfill $Cond_1$ and count the *tagging* behavior at path index i. We increase $t[A_i], if A_i{:}* \in output(A_1)$. Otherwise, we increase $s[A_i]$.
  - **count *forwarding*** We use *forwarding* and *tagging* counters from previous steps to fulfill $Cond_1$ and $Cond_2$ and count the *forwarding* behavior at path index i. We increase $i[A_i], if A_t{:}* \in output(A_1)$. Otherwise, we increase $c[A_i]$.

Summary: In essence, the inference method iterates over the input list L two times, i.e., $2 * (N * |L|)$, where N corresponds to the maximum path index. For each path index (column) we first count the *tagging* behavior, then the *forwarding* behavior. With each iteration over a path index, the gained knowledge is transferred to the next iteration. We note that at high path indices the number of ASes that need to fulfill $cond_1$ increase, and thus, the probability of inference decreases. Although the maximum observed path length in the input data is 19 (after cleaning), we observe that the algorithm stops increasing counters, e.g., due to missing information, typically at around N=7.

5.7 Discussion

In order to apply rules that require pre-existing knowledge and at the same time traverse the input data efficiently, we chose an approach that considers both. Instead of traversing the entire paths one by one (row-based), in our approach we traverse all paths by the path index (column-based), starting with the collector peer. Recall, $output(A_1)$ is recursive and thus can contain the community output of other ASes along the path.

Row-based: In a row-based approach, each path is processed separately and independently from the other paths. Thus, the counters for observing communities (or the lack thereof) associated with an AS path cannot be connected to observations in other paths. Inferences can be made based on the counters only after all paths are processed. In order to achieve our objectives, i.e., classifying consistent behavior with high precision, additional refinement and optimization steps would be required.

Further, since the row-based approach counts observations without pre-existing knowledge about the ASes in the AS path, it is susceptible to misclassification, e.g. due to noise or hidden behavior. Also, since there is no ground truth information available about the community usage in the wild, finding the right threshold to adjust the precision is not possible. In the appendix, we provide pseudo-code that serves as an example for a row-based approach, see Listing 2.

Column-based: Like the row-based approach, our column-based approach allows us to count while we traverse the AS paths. However, in addition it can generate pre-existing knowledge at one path index and used it at later indices. This further enables us to implement rules, i.e. $Cond_1$ and $Cond_2$ that base on our mental model, on every AS path. Thus, our algorithm is able to identify

- ASes which behavior is hidden behind a *cleaner* AS and
- ASes that are not illuminated by a *downstream tagger*.

Both abilities allow us not only to minimize misclassification, but also they increase the robustness of inferences in the presence of noise. In Section 6 we show that our algorithm avoids misclassifications and that it classifies less than 0.5% of ASes that are actually hidden.

6 VERIFICATION OF ALGORITHM

In this section, we apply the algorithm to different scenarios in controlled simulations, in order to verify that it performs according to our expectations. We generate multiple data sets with known community usage behavior. Therefore, we take all available AS paths from RIPE, RouteViews, and Isolario (d_{May21}), assign roles to each AS, e.g., *tf* (*tagger-forward*), *sc* (*silent-cleaner*), and compute the community output of each peer AS ($output(A_1)$), according to our mental model in Section 3.3. Thus, we have an AS path substrate from real-world BGP data augmented with known community usage behavior per AS. Specifically, we know which AS's behavior is consistent (*tagger*, *forward*, ...) and selective and which behavior is always hidden and thus cannot be inferred. Further, each scenario uses the AS paths from d_{May21} as substrate. Thus, each generated data set includes 72,951 ASes among which 60,420 are leaf ASes and 766 are collector peers. All presented results were inferred using a threshold of 99% in order to increase the inference counters.

6.1 Consistent behavior

To test our algorithm against the average and extreme cases, we generate ground truth data sets with different settings in the community usage behavior of ASes. Initially, we consider consistent behavior of ASes, e.g., a *tagger* adds communities towards all BGP neighbors (including collectors), irrespective of the business relationship.

- **alltf:** To test how the algorithm performs under optimal conditions, in this scenario we maximize the visibility of communities by assigning the role *tf* to all ASes. This way the *tagging* and *forwarding* behavior of all ASes becomes visible.
- **alltc:** In this scenario we test the opposite case, i.e., where the visibility of communities is minimized. Therefore, the role *tc* is assigned to all ASes.
- **random:** Since there exist no ground truth information regarding the community usage of ASes, we create an average case scenario, where all roles *tf*, *tc*, *sf*, and *sc* are uniformly assigned at random among all ASes, i.e., 25% are *tf*, 25% *tc*, 25% *sf*, and 25% *sc*.

Intuitively, the algorithm should perform best in the *alltf* scenario and perform worst in the *alltc* scenario. Note, since each AS behaves consistently, the *tagging* and *forwarding* behavior, if not hidden, will be inferred correctly, irrespective of the threshold.

In order to better understand the sensitivity of our inference method, we add additional communities to the generated ground truth data sets that stress the ability of the algorithm to infer the *tagging* and *forwarding* behavior of ASes correctly.

- **noise:** First, to test the *tagging* behavior inference, we let around 50% of all ASes set communities using the ASN of their *upstream* neighbor in the *upper* field, simulating action communities. Second, to test the *forwarding* inference, we add a community with the *upper* field corresponding to the ASN of the originator.

These two noise sources occur with a 5% probability, such that each affected AS can exhibit inconsistent behavior. In the next section, we introduce additional scenarios in which we explore the *selective* behavior of ASes.

6.2 Selective behavior

Since not all ASes exhibit a consistent community usage behavior, their *tagging* and *forwarding* counters may lead to an *undecided* inference, or even to a misclassification, see Section 5.4. In order to test how the algorithm reacts to *selective* behavior, we use the *random* scenario with consistent behavior and modify around 50% of the assigned *tagger* ASes to selectively tag routes based on the business relationship:

- **random-p:** In this scenario, the *selective tagger* ASes do not set communities on provider links, but on all other links, i.e., peers, customers and collectors.
- **random-pp:** In this scenario, the *selective tagger* ASes do not set communities on provider and peer links; only towards and customers and collectors.

For our convenience, we use CAIDAs business relationship inferences [3] to augment the ground truth data sets with the corresponding behavior. For example, a *selective tagger* AS A_x in scenario *random-pp* adds a community towards its neighbor A_{x-1}, only if A_{x-1} is a customer of A_x. Since, we create a hypothetical scenario, we do not rely on the accuracy of the business relationship inferences.

Other than that, the parameters of the *random* scenario of Section 6.1 are used. Since in *random-p* and *random-pp* the propagation of communities is limited, the algorithm is expected to perform worse compared to the *random* scenario.

	tagging		*forwarding*		full classification				partial classification				none / undecided			
scenario	rec.	prec.	rec.	prec.	*tc*	*sc*	*tf*	*sf*	*tn*	*sn*	*nc*	*nf*	*nn*	u*	*u	uu
consistent behavior:																
alltc	1.00	1.00	0.82	1.00	578	0	0	0	188	0	0	0	72,185	0	0	0
alltf	0.96	1.00	0.83	1.00	0	0	10,427	0	59,570	0	0	0	2,954	0	0	0
random	0.93	1.00	0.70	1.00	1,295	1,298	1,319	1,295	20,558	20,613	0	0	26,573	0	0	0
random+noise	0.55	1.00	0.45	1.00	459	242	1,256	618	20,067	3,282	1	1	27,808	17,518	1,288	412
selective behavior:																
random-p	0.42	0.86	0.39	0.97	623	766	403	482	7,275	13,513	1	1	48,980	622	270	16
andom-pp	0.18	0.89	0.18	0.94	329	399	134	169	3,484	5,817	0	1	62,035	286	288	12

Table 2: Classification results and performance using scenarios with consistent and *selective* behavior. The numbers represent mean values from 10 iterations per *random* scenario. Thresholds 99%, ground truth data substrate: d_{May21}, 72,951 ASes (60,420 leafs, 31,239 32-bit, 766 collector peers), in May 19, 2021.

6.3 Results

In Table 2 we present the classification results for the scenarios with consistent behavior (*random*, *alltf*, *alltc*), and those with *selective* behavior (*random-p*, *random-pp*). In each *random* scenario, we use 10 different ground truth data sets, each generated with a different (random) distribution of roles (tf, tc, sf, sc). Since, the resulting inferences are comparable in all 10 variations, we only show the mean values in the table.

For simplicity, we calculate the recall by considering only *tagging* and *forwarding* behaviors that are not selective, hidden or missing (i.e., *forwarding* behavior at leaf ASes). Thereby, the false negatives include cases where the *tagging* or *forwarding* behavior was not inferred, i.e., *none*, or the inference results in *undecided*.

Let us first look at scenarios with consistent behavior:

- **Precision and recall:** All scenarios with consistent behavior (including random+noise) show a precision of 100%, i.e., there is no wrong inference of *tagging* and *forwarding* behavior. Also, the recall is relatively high when inferring *tagging* (93-100%) and *forwarding* (70-82%). However, the recall in the case of random+noise is low, i.e., 55% for *tagging* and 45% for *forwarding* behavior.
- ***full classification*:** In the *random* scenario, the algorithm yields around 1,300 inferred ASes for each of the assigned roles, i.e., both, the *tagging* and *forwarding* behaviors were inferred. Further, in the *alltc* scenario 578 ASes are *tc* and in the *alltf* scenario 10,427 ASes are tf.
- ***partial classification*:** A significant amount of ASes have an inferred *tagging* behavior but no *forwarding* behavior. ~20K ASes are identified as *tagger* and ~20K as *silent* in the *random* scenario.
- ***none / undecided*:** Looking at the number of not classified ASes, the algorithm performs best in the *alltf* scenario, and worst in the *alltc* scenario, as expected. Notably, the random+noise scenario leads to *undecided* behavior in particular the *tagging* behavior is affected, see u* (*tagging* is *undecided*).

Next, we look at the *selective* behavior of ASes, i.e., *random-p* and *random-pp*. Compared to the simple *random* scenario, the *selective* behavior of ASes lead to an increase of *undecided* inferences in the case of *tagging* and *forwarding*, see u* (*tagging* is *undecided*) and *u (*forwarding* is *undecided*) in Table 2. Note that *undecided* behavior exacerbates the inference of other ASes, see Section 5.4.

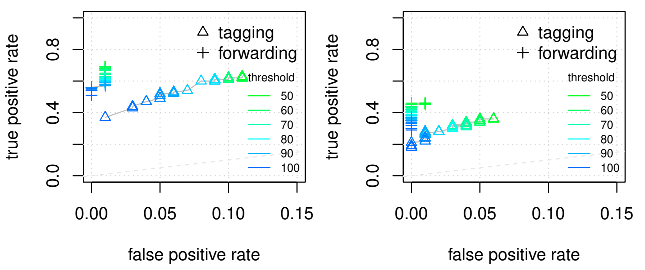


Figure 2: ROC curves show effects when changing threshold. In scenario *random-p* (left) *taggers* add communities on customer and peering links, and in scenario *random-pp* (right) *taggers* add communities on customer links only (increased difficulty). Performance not sensitive to threshold.

Unsurprisingly, the distribution of inferences is skewed compared to the simple *random* scenario. Also, when we consider the *none* cases as a performance metric, we find that the algorithm still performs better than the *alltc* scenario and worse than the *alltf* scenario.

The most important difference resulting from those scenarios is the recall and precision. Because of *selective taggers* the precision of our inference method in both scenarios is affected, i.e., 86%/99% in the case of *tagging* and 97%/94% in the case of *forwarding*. The recall is more affected: 42%/18% in the case of *tagging* and 39%/18% in the case of *forwarding*. Given the increased difficulty in the *random-p* and *random-pp*, these results are expected.

6.3.1 Performance under different thresholds. Selective *tagging* behavior can lead to a contradicting perception of community usage, i.e., the *tagger* counter $t[Z]$ for a given AS Z is increased in one instance, the *silent* counter $s[Z]$ for the same AS may be increased in a another. In the following, we show how the sensitivity and specificity of our inferences change when different thresholds are used to infer the *tagging* or *forwarding* behavior of an AS. We repeat the inference method on both scenarios, *random-p* and *random-pp*, for every threshold between 50% and 100%.

Figure 2 show the ROC curves for the *tagging* and the *forwarding* inferences, for the scenarios *random-p* (left) and *random-pp* (right), respectively. The false positive rate is on the x-axis and the true positive rate on the y-axis. As we can see, by increasing the threshold from 50% to 100% the specificity increases and the sensitivity decreases. However, those are minute changes: The false positive rate decreases from 10% to 1% in case of the *tagging* classifier, and from 1% to 0% in case of the *forwarding* classifier. These results indicate that the inferences are not very sensitive to the threshold, in particular when inferring the *forwarding* behavior. Also, the true positive rate decreases by ~20% in both cases. Since the difficulty in scenario *random-pp* is increased, the number of undecided ASes increase as well. Thus, we observe a lower true positive rate compared to *random-p*.

6.4 Scenario Impact

In this section, we explain the impact of the different scenarios on the inference of *tagging* and *forwarding* behavior, respectively. Thereby, we contrast the assigned roles against classification results with the help of confusion matrices, as provided in the tables below. Where necessary, we also include the information *(hidden)* if the assigned role is hidden (Section 5.1.2), and *(leaf)* if the *forwarding* behavior is assigned to a leaf AS (Section 5.1.3). Recall that each scenario uses the AS paths from d_{May21} as substrate which includes 72,951 ASes, among which are 60,420 leaf ASes and 766 collector peers. In the following we provide a selection of results. For a complete list, see Tables 5 and 6 in the appendix.

alltf: Let us consider the scenario *alltf* where all ASes are assigned the role *tagger-forward* and the visibility is maximized:

assigned roles:	classification result:			
alltf:	*tagger*	*silent*	*undecided*	none
tagger	69,997	0	0	2,954

The algorithm is able to attribute a correct *tagging* behavior to more than 95% of the ASes. For the remaining 2,954 ASes, the algorithm cannot generate pre-existing knowledge for neighboring ASes due to a race condition; A limitation of our algorithm as described in Section 5.2.1. Thus, there are no counters and the inference returns *none*.

When we look at the *forwarding* behavior in the same scenario, we see that more than 60K ASes are leaf ASes, and are therefore not attributed a *forwarding* behavior:

assigned roles:	classification result:			
alltf:	*forward*	*cleaner*	*undecided*	none
forward	10,427	0	0	2,104
forward (leaf)	0	0	0	60,420

Given those ASes that are not leaf, the algorithm is able to correctly classify more than 83% (10,427) as *forward*.

alltc: In this scenario, all ASes are *tagger-cleaner*. Since all ASes are *cleaner*s, we correctly infer the *tagging* behavior of all 766 collector peers:

assigned roles:	classification result:			
alltc:	*tagger*	*silent*	*undecided*	none
tagger	766	0	0	0
tagger (hidden)	0	0	0	72,185

The remaining ~72K ASes are hidden behind the those peers and thus fall into *none*. Next, we look at the *forwarding* behavior in the same scenario:

assigned roles:	classification result:			
alltc:	*forward*	*cleaner*	*undecided*	none
cleaner	0	578	0	124
cleaner (hidden)	0	0	0	11,829
cleaner (leaf)	0	0	0	60,420

From the 766 *tagger* collector peers, 578 are followed by other collector peers in the AS-level graph. Recall, the algorithm first determines the *tagging* behavior of all peers, and then uses them as *downstream tagger*s to determine their *forwarding* behavior. 124 peer ASes (as well as 11,829 transit ASes) are never followed by another peer AS, thus they fall into *none*. The remaining 64 peer ASes appear as leaf and are included in the set of 60,420 leaf ASes.

random: Next, we look at the *tagging* behavior in the random scenario where, due to lack of ground truth, we assign roles uniformly at random. Since there are 72,951 ASes in the data set, each *tagging* role (*tagger*, *silent*) occurs approximately ~36K times:

assigned roles:	classification result:			
random:	*tagger*	*silent*	*undecided*	none
tagger	22,149	0	0	1,541
silent	0	21,966	0	1,571

skipped hidden

Given the distribution of roles in this scenario, around ~13K *tagger* ASes and ~13K *silent* ASes are hidden behind a *cleaner*. From the remaining ~23K ASes, more than 90% (~22K) are correctly assigned to the respective class, while for ~1.5K ASes the algorithm fails to generate pre-existing knowledge.

random+noise: Thus far, our algorithm is able correctly infer the *tagging* and *forwarding* behavior of ASes and avoids (mis)classifying hidden ASes. In the following, we show how the inferences change when the same random scenario (same seed to randomly assign roles) is augmented with noise:

assigned roles:	classification result:			
random+noise:	*tagger*	*silent*	*undecided*	none
tagger	21,625	0	1	2,064
silent	53	3,679	17,687	2,118
tagger (hidden)	0	12	2	12,766
silent (hidden)	1	9	3	12,931

Most notably, >80% (17,687) of the *silent* ASes are affected by the addition noise such that they are labeled *undecided*. One the other hand, *tagger* ASes are minimally affected compared to the random scenario (524 correct inferences less). Also, we observe 53 misclassifications and classifications where the behavior is hidden.

Similar to *silent* ASes in the case of *tagging* behavior, *cleaner* ASes are also affected by noise:

assigned roles:	classification result:			
random+noise:	*forward*	*cleaner*	*undecided*	none
forward	2,294	0	63	1,134
cleaner	1	738	1,647	1,153

skipped hidden and leaf

Here, 1,647 (or ~68%) of *cleaner* ASes are labeled *undecided* compared to the random scenario.

random-p: Lastly, we consider the random scenario where around 50% of the *tagger* ASes are randomly chosen to tag selectively, i.e., not on provider links but on all other links (including collectors). This results in the following distribution of *tagging* roles: *selective* ~18K, *tagger* ~18K, and around 36K *silent* ASes.

assigned roles: random-p:	classification result: *tagger*	*silent*	*undecided*	none
tagger	6,750	0	0	5,876
silent	0	13,445	0	11,983
selective	2,351	3,538	837	5,984
skipped hidden				

In general, the number of not inferred ASes (*none*) increases in all cases compared to the random scenario. Since *selective* ASes are not tagging towards their providers fewer communities make it to the collector. Regarding the visible *tagger* and *silent* ASes, there are no misclassifications. However, almost 50% of the ASes in both cases are not inferred. Interestingly, from the ~12K visible *selective* ASes, around 5.9K are classified either *tagger* or *silent*. Only 837 ASes are *undecided*.

Summary: The algorithm classifies consistent behavior with a high precision. It avoids misclassifications of *silent* in presence of noise (*undecided* instead), while *tagger* ASes are mostly unaffected. Furthermore, the algorithm avoids classifying ASes that are hidden or leaf ASes.

7 ANALYSIS

In the previous section, we have introduced an inference method that allows us to infer the *tagging* and *forwarding* behaviors of ASes. We devised an algorithm that utilizes routing information, specifically AS paths and community attributes, and validated its properties using customized data sets.

Next, we show the classification results when the inference method is applied on actual, unmodified routing data, characterize the involved ASes, and validate our inferences by actively injecting route announcements with attached communities to the real-world routing system.

7.1 Classification results

Table 3 summarizes the community usage classification per route collector project and of the aggregation of RIPE, RouteViews, and Isolario (d_{May21}). We also include the inferences based on PCH update data only. Note that the PCH data set does not include RIB information, see Section 4. We split the statistics by *tagging* and *forwarding*. The algorithm exhibits a similar share of identified behavior among the three collector projects RIPE, RouteViews, and Isolario: 717 to 809 ASes exhibit *tagger* behavior, 9,901 to 10,594 *silent*, 198 to 216 *forward*, and 239 to 309 ASes exhibit *cleaner* behavior. Looking at the aggregate (d_{May21}) the numbers are comparable (860, 12,315, 271, and 417, respectively). PCH shows the least amount of inferred classes.

Regarding the full classification, i.e., both, the *tagging* and the *forwarding* for an AS are inferred, we see that *tf* (*tagger-forward*), *tc* (*tagger-cleaner*), and *sf* (*silent-forward*) show similar numbers; between 47 and 88. *sc* (*silent-cleaner*) is the most common with 129 to 173 inferred ASes. The aggregated data set shows an increase

Input data	RIPE	RouteViews	Isolario	d_{May21}	PCH
tagging:					
tagger	723	717	809	860	521
silent	10,594	9,902	9,901	12,315	2,214
undecided	778	827	902	994	286
none	60,642	61,391	61,010	58,782	64,520
forwarding:					
forward	201	198	216	271	173
cleaner	285	309	239	417	133
undecided	216	137	146	308	85
none	72,035	72,193	72,024	71,995	67,150
tagger-forward	51	70	79	84	68
tagger-cleaner	58	64	47	81	37
silent-forward	88	66	83	107	72
silent-cleaner	167	173	129	251	70

Table 3: Classification results using real BGP data (RIBs + updates), May 19, 2021. RIPE, RouteViews, and Isolario are comparable. Aggregated data set d_{May21} yields most classifications. PCH includes only updates.

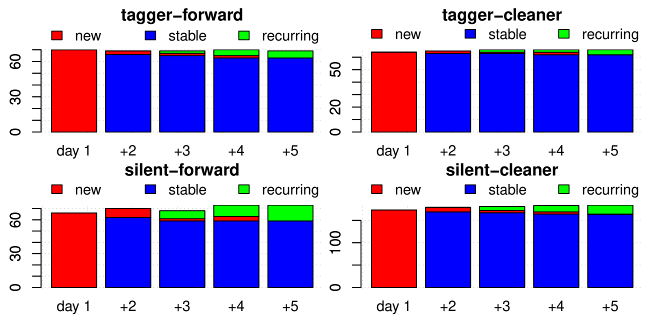


Figure 3: Impact of incrementally adding more days to the input data. For each full classification (*tf*, *tc*, *sf*, *sc*), barplot shows the total number of ASes (y-axis) and the number of successive days used (x-axis). *New* ASes appear for the first time in the respective full class, *stable* ASes appear consistently since day 1 (May 19, 2021), and *recurrent* ASes occur with interruptions. RouteViews data only.

compared to the individual collectors with a total of 523 fully classified ASes out of which 273 are collector peers. Next, we take a closer look at the stability and longitudinal changes of our inferences.

7.1.1 Stability and longitudinal view. In order to better understand how adding more data impacts our inferences, we next apply our algorithm to a day worth of data which we incrementally increase by a successive day for a total of 5 days, starting in May 19, 2021. Thus, the final data set from May 23 contains data from all 5 days. Since the inference numbers in Table 3 for the collectors and the aggregated d_{May21} are comparable, in the following we focus only on RouteViews data sets.

Figure 3 shows the number of *new*, *stable*, and *recurrent* ASes for each full class (*tf*, *tc*, *sf*, *sc*). We can see that except for day 1, only few ASes are new, i.e., a maximum of 10 at day +2 in the case of *sc*. Moreover, while we observe some amount of recurring ASes, the majority (90-97%) of ASes appear stable since day 1. We are

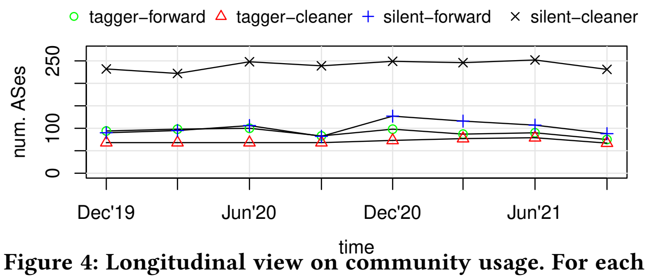


Figure 4: Longitudinal view on community usage. For each full classification (*tf*,*tc*,*sf*,*sc*), number of total ASes (y-axis) per day plotted over a time period of 2 years, from December 15, 2019 to (and including) September 15, 2021, every three months. Aggregated data from RIPE, RouteViews, and Isolario used.

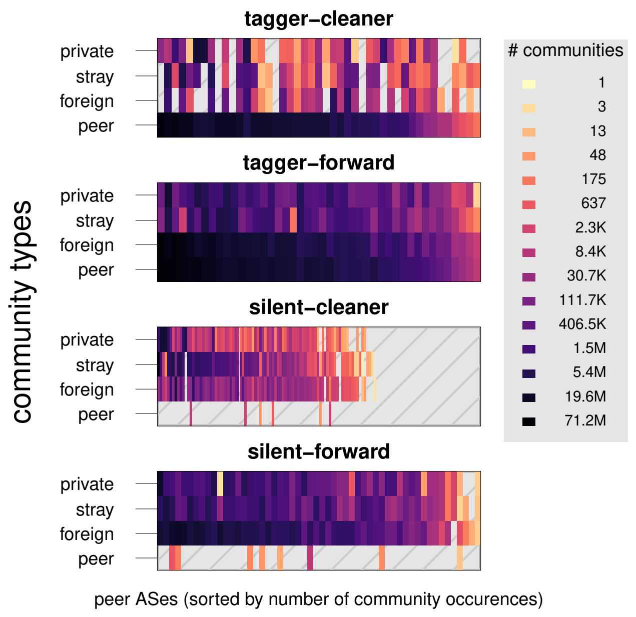


Figure 5: Counting different community types (y-axis) at peer ASes (x-axis) with full classification, i.e., *tf*, *tc*, *sf*, and *sc*. The ASes are ordered by number of communities and the color scale is logarithmic.

confident that using our algorithm a day worth of routing data is of sufficient granularity to provide stable inferences.

Next, we investigate how the number of fully classified ASes (*tf*, *tc*, *sf*, and *sc*) change over time. Therefore we use aggregated data sets, i.e., RIPE, RouteViews, and Isolario combined, one day every three months over a time period of two years, see Figure 4. The time period ranges from December 15, 2019 to (and including) September 15, 2021. There is no significant increase or decrease in the number of fully classified ASes discernible. Rather, the number of individual ASes per class and per day are comparable to those presented in Table 3 for d_{May21}. Overall, these results are indicative of a stable, consistent community usage behavior throughout the past two years, involving a small set of ASes.

7.2 Peer ASes

In the following, we take a closer look at 273 collector peers of the 523 fully classified ASes, see Section 7.1. Since peer ASes are connected directly to BGP collectors, their community set output ($output(A_1)$) should be not impaired by other ASes along the path. Specifically, we investigate whether $output(A_1)$ corresponds to the inferred *tagging* and *forwarding* behavior of a peer AS A_1. For that, we count the type of communities that are included in $output(A_1)$, i.e., *peer*, *foreign*, *stray*, and *private*, see Section 3.2.

Based on our mental model in Section 3.3, the intuition of our approach is as follows: *peer* communities should only be observable when the peer AS is a *tagger*. We should not be able to observe *peer* communities, when the AS is *silent*. On the other hand, *foreign* should only be visible if the peer AS is *forward*, and not visible if that AS is a *cleaner*. Note, since our inference method discards *stray* and *private* communities (Section 5.1), we expect to see them independent of the peer AS classification.

Figure 5 shows four scatter plots, one for each (full) class *tf* (*tagger-forward*), *tc* (*tagger-cleaner*), *sf* (*silent-forward*), and *sc* (*silent-cleaner*). In each plot, the x-axis contains peer ASes of the corresponding class, ordered by the number of total communities, and the y-axis indicates the community types. The color palette indicates the number of total communities in logarithmic scale, ranging from 1 to 71M+.

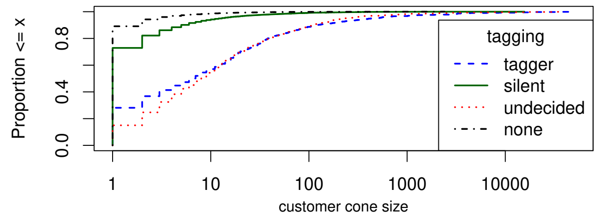
In accordance with our expectations, we observe a large amount *peer* communities in the cases *tf* and *tc*, indicated by the bottom bar of the respective plots. We observe few to no *peer* communities in the cases *sf* and *sc*. Furthermore, we observe a large amount of *foreign* communities in the cases *tf* and *sf*, i.e., when the peer ASes does not remove communities from other ASes in the AS path (see second bottom bar). Conversely, we see only few to no *foreign* communities when the peer AS is *tc* or *sc*, i.e., the AS is removing communities form other ASes in the AS path. However, there are also some *cleaner* ASes that, given the number of *foreign* communities, contradict their inference. Those *foreign* communities can be caused, e.g., by one or multiple *downstream* ASes that our algorithm was not able to identify as *tagger* (and thus invalidate $Cond_2$). Last, Since we do not consider *stray* and *private* communities in our inference method, we observe them in all classes, in particular when the peer AS is *forward*, but also when it is a *cleaner*.

Conclusion: Overall, our observations of community usage at collector peers align with our mental model and assure the correct functioning of our inference method.

7.3 AS characterization

Next, we explore the characteristics of ASes based on their classification. For that we make use of CAIDAs customer cone data sets [3]. The customer cone of an AS includes itself and all ASes that can be reached by only traversing customer links. Leaf ASes thus have a customer cone size of 1. We use the customer cone size as indicator for the size of an AS.

In Figure 6, we plot CDFs over customer cone sizes over all ASes in d_{May21}. Starting with the *tagging* behavior (top plot), we see a significant difference of cone sizes between ASes that are classified as *tagger* and those that are classified as *silent*. ~70% of *silent* ASes have a cone size of 1, i.e., they are leaf ASes. Also, while around 10%



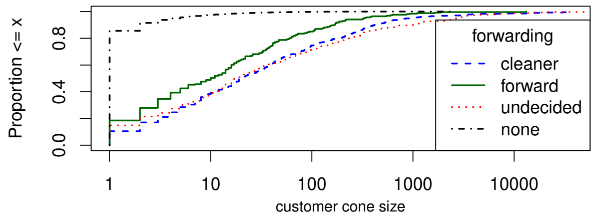


Figure 6: CDFs showing customer cone size distribution of ASes per *tagging* behavior (top) and *forwarding* behavior (bottom). All behaviors (except for *silent*, and those that are *none*) are attributed to mostly larger and non-leaf ASes.

experiment date	communities present	communities not present
2021-05-19	6/177 (3%)	285/367 (78%)
2021-07-15	1/104 (1%)	286/365 (78%)
2021-08-15	0/61 (0%)	300/359 (84%)

Table 4: PEERING experiments: Table shows the share of paths containing at least one *cleaner* AS, when a) communities are present and b) when communities are not present. We perform three temporally uncorrelated experiments.

of *silent* ASes have a cone size greater than 10, it is around 50% for *tagger* ASes, indicating that ASes at the edge of the AS level graph usually do not add their own communities. Interestingly, ASes with *undecided* behavior exhibit a similar distribution of cone sizes as *taggers*. Lastly, ASes with no inference are mostly leaf ASes, i.e. in ~90% of the cases. Most notably for *forwarding* behavior (bottom plot), *cleaner* and *silent* ASes show a similar distribution of cone sizes, indicating that *cleaner* and *silent* ASes are common across all AS sizes. Again, ASes with no inference are mostly leaf ASes.

7.4 Validation of inferences

Finally, we validate our inferences with external ground truth information obtained by injecting route announcements with consistent use of communities. Thus, by having control over a *tagger*, we check how often the observation of our communities contradicts the inferences of AS behaviors along the corresponding AS path. For example, given an AS path and a community set that include our communities, the AS path must not include a *cleaner* AS, otherwise the community would have been not visible (see Section 5.1). Similarly, if the community set does not include any of our communities, there must be at least one *cleaner* in the AS path.

We utilized the PEERING testbed to conduct our real-world validation [23]. Using PEERING, we announce a /24 prefix via 12 available Points of Presence (PoPs) for approximately one week beginning on May 19, 2021. To each PoP, we add a unique pair of communities to the announcement; these communities use PEERING's 47065 ASN in the upper 2 bytes and unique values in the lower 2 bytes. Via these PoPs, PEERING had 460 active BGP peers in total at the time of our experiment.

We extract the relevant announcements from our data set d_{May21} by filtering for the prefix. We observe a total of 7,503 announcement corresponding to that prefix. From the 12 community sets we have configured, only 6 appear in around 30% of those announcements. From the observed announcements, we extract 549 unique AS paths. Interestingly, only 5 AS paths are not consistently attached with our communities, i.e., they appear with and without out communities. The remaining 544 AS paths are either always associated with our communities or never. This indicates a consistent usage of communities along the same path.

In order to validate our inferences, we count how often a *cleaner* exists in the AS paths, when a) the corresponding community set contains our communities and when b) the community set *does not* contain our communities. We first look at 367 unique tuples that do not include any of our communities: in 285 cases (~78%) at least one *cleaner* exists in the AS path. Further 81 cases (~22%) include at least one AS that exhibits *undecided* behavior. Only 1 AS path does not include an identified *cleaner* and thus contradicts our inferences. Next, we select 177 unique AS path / community set tuples that *do* contain our communities: Here, 6 (3%) AS paths contradict our inferences, i.e., the AS path contains a *cleaner* AS. However, we note that 152 AS paths (~85%) include at least one AS with *undecided* behavior. The remaining 19 AS paths include neither. In Table 4 we show a summary of this and two additional experiments we have performed, with similar outcome.

8 CONCLUSIONS AND FUTURE WORK

In this work, we take a step toward achieving a more grounded understanding of BGP community usage behavior at a per-AS granularity. While our algorithm is inferential and its ability to classify an AS's community usage depends both upon BGP visibility and the complexity of the AS's policies, ours is the first work to take on this challenging task. Furthermore, through extensive testing and validation, we show that our algorithm has high precision – when it does make an inference, it is generally correct.

We apply our algorithm to large-scale, real-world BGP data from the major route collectors in order to maximize coverage and make inferences for the most ASes possible. We characterize the ASes exhibiting different usage behaviors and find that *tagger*, *forward* and *cleaner* ASes typically have a large customer cone, while *silent* are typically at the edge.

We publicly release our algorithm and inferences to the community to support related BGP and network research, including automated community processing in networks based on inferences, Internet modeling, and improved BGP security posture [5].

In future work, we plan to extend and improve the algorithm, especially removing the strict assumption that community tags use the ASN of the network that added the community. In this fashion, we wish to identify not only whether an AS is a *tagger*, but also *which* communities it adds. This ability will be especially useful to differentiate signaling versus informational community, and to support efforts directed toward automated and safe community filtering.

Acknowledgements

We would like to thank our shepherd, Neil Spring, and the anonymous reviewers for useful and constructive feedback. We would also like to thank Italo Cunha and Ethan Katz-Bassett for providing access to the PEERING test bed. This work has been partially funded by NSF grant CNS-1855614, the European Research Council (ERC) Starting Grant ResolutioNet (ERC-StG-679158), BMBF BIFOLD 01IS18025A and 01IS18037A, and performed while the first author held an NRC Research Associateship award at the Naval Postgraduate School. Views and conclusions are those of the authors and should not be interpreted as representing the official policies or position of the U.S. government, the NSF, ERC, BMBF, or NRC.

REFERENCES

[1] Larry Blunk, Craig Labovitz, and Manish Karir. 2011. Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format. RFC 6396. (Oct. 2011). https://rfc-editor.org/rfc/rfc6396.txt

[2] Jay Borkenhagen, Randy Bush, Ron Bonica, and Serpil Bayraktar. 2019. Policy Behavior for Well-Known BGP Communities. RFC 8642. (Aug. 2019). https://doi.org/10.17487/RFC8642

[3] CAIDA. 2021. AS Relationships. https://www.caida.org/catalog/datasets/as-relationships/. (2021).

[4] CISCO. 2005. Remotely Triggered Black Hole Filtering - Destination Based and Source Based. Cisco White Paper, http://www.cisco.com/c/dam/en_us/about/security/intelligence/blackhole.pdf. (2005).

[5] CMAND. 2021. AS-Level BGP Community Usage Classification. https://www.cmand.org/communityusage/. (2021).

[6] Amogh Dhamdhere and Constantine Dovrolis. 2011. Twelve years in the evolution of the internet ecosystem. *IEEE/ACM Transactions on Networking* 19, 5 (2011), 1420–1433.

[7] Benoit Donnet and Olivier Bonaventure. 2008. On BGP Communities. In *ACM SIGCOMM CCR.* https://doi.org/10.1145/1355734.1355743

[8] Vasileios Giotsas, Christoph Dietzel, Georgios Smaragdakis, Anja Feldmann, Arthur Berger, and Emile Aben. 2017. Detecting Peering Infrastructure Outages in the Wild. In *ACM SIGCOMM.* https://doi.org/10.1145/3098822.3098855

[9] Vasileios Giotsas, Matthew Luckie, Bradley Huffaker, and k claffy. 2014. Inferring Complex AS Relationships. In *ACM IMC.* https://doi.org/10.1145/2663716.2663743

[10] Vasileios Giotsas, George Nomikos, Vasileios Kotronis, Pavlos Sermpezis, Petros Gigis, Lefteris Manassakis, Christoph Dietzel, Stavros Konstantaras, and Xenofontas Dimitropoulos. 2020. O Peer, Where Art Thou? Uncovering Remote Peering Interconnections at IXPs. In *IEEE/ACM ToN.*

[11] Vasileios Giotsas, Georgios Smaragdakis, Christoph Dietzel, Philipp Richter, Anja Feldmann, and Arthur Berger. 2017. Inferring BGP Blackholing Activity in the Internet. In *ACM IMC.* https://doi.org/10.1145/3131365.3131379

[12] Timothy G Griffin and Gordon Wilfong. 1999. An Analysis of BGP Convergence Properties. In *ACM SIGCOMM CCR.* https://doi.org/10.1145/316194.316231

[13] Jakob Heitz, Job Snijders, Keyur Patel, Ignas Bagdonas, and Nick Hilliard. 2017. BGP Large Communities Attribute. RFC 8092. (Feb. 2017). https://rfc-editor.org/rfc/rfc8092.txt

[14] Isolario. 2021. Isolario project. https://www.isolario.it/. (2021).

[15] Thomas Krenc, Robert Beverly, and Georgios Smaragdakis. 2020. Keep your Communities Clean: Exploring the Routing Message Impact of BGP Communities. In *CoNEXT.*

[16] Tony Li, Ravi Chandra, and Paul S. Traina. 1996. BGP Communities Attribute. RFC 1997. (Aug. 1996). https://rfc-editor.org/rfc/rfc1997.txt

[17] Vern Paxson. 1996. End-to-End Routing Behavior in the Internet. In *ACM SIGCOMM.* https://doi.org/10.1145/248156.248160

[18] PCH. 2021. Packet Clearing House. https://www.pch.net/. (2021).

[19] Yakov Rekhter, Susan Hares, and Tony Li. 2006. A Border Gateway Protocol 4 (BGP-4). RFC 4271. (Jan. 2006). https://rfc-editor.org/rfc/rfc4271.txt

[20] Philipp Richter, Georgios Smaragdakis, Anja Feldmann, Nikolaos Chatzis, Jan Boettger, and Walter Willinger. 2014. Peering at Peerings: On the Role of IXP Route Servers. In *ACM IMC.* https://doi.org/10.1145/2663716.2663757

[21] RIPE. 2021. RIS - RIPE Network Coordination Centre. http://ris.ripe.net/. (2021).

[22] RouteViews. 2021. University of Oregon RouteViews project. http://www.routeviews.org/. (2021).

[23] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. 2019. PEERING: Virtualizing BGP at the Edge for Research. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies.* 51–67.

[24] Pavlos Sermpezis, Vasileios Kotronis, Petros Gigis, Xenofontas Dimitropoulos, Danilo Cicalese, Alistair King, and Alberto Dainotti. 2018. ARTEMIS: Neutralizing BGP hijacking within a minute. *IEEE/ACM Transactions on Networking* 26, 6 (2018), 2471–2486.

[25] Job Snijders, John Heasley, and Martijn Schmidt. 2017. Use of BGP Large Communities. RFC 8195. (June 2017). https://rfc-editor.org/rfc/rfc8195.txt

[26] Florian Streibelt, Franziska Lichtblau, Robert Beverly, Anja Feldmann, Cristel Pelsser, Georgios Smaragdakis, and Randy Bush. 2018. BGP Communities: Even more Worms in the Routing Can. In *ACM IMC.* https://doi.org/10.1145/3278532.3278557

[27] Cecilia Testart, Philipp Richter, Alistair King, Alberto Dainotti, and David Clark. 2019. Profiling BGP Serial Hijackers: Capturing Persistent Misbehavior in the Global Routing Table. In *Proceedings of the Internet Measurement Conference (IMC '19).* Association for Computing Machinery, New York, NY, USA, 420–434. https://doi.org/10.1145/3355369.3355581

[28] Quaizar Vohra and Enke Chen. 2012. BGP Support for Four-Octet Autonomous System (AS) Number Space. RFC 6793. (Dec. 2012). https://doi.org/10.17487/RFC6793

A APPENDIX

```
for x = 1, ..., N:
  # PHASE 1: count tagging
  for each ($path, comm$):
    assert(cond1)
    t[$A_x$]++ if $A_x$:* in comm
    s[$A_x$]++ if $A_x$:* not in comm
  # PHASE 2: count forwarding
  for each ($path, comm$):
    assert(cond1 and cond2)
    i[$A_x$]++ if $A_t$:* in comm
    c[$A_x$]++ if $A_t$:* not in comm
```

Listing 1: Inference algorithm, iterating over $(path, comm)$ pairs by path index x, where $path = A_1, A_2, ..., A_n$, and $comm = output(A_1)$.

```
# INIT
aslist[] = silent | forward

# PHASE 1: count tagging
for each ($path, comm$):
  for each i = 1, ..., n:
    t[$A_x$]++ if $A_x$:* in comm
    s[$A_x$]++ if $A_x$:* not in comm
# PHASE 2: count forwarding
for each ($path, comm$):
  for each x = n − 1, ..., 1:
    c[$A_x$]++ if $A_{x+1}$ not in comm
    else:
      i[$A_j$]++ for j in 1, ..., $A_{x+1}$
```

Listing 2: Alternative inference algorithm, iterating over $(path, comm)$ pairs without conditions, where $path = A_1, A_2, ..., A_n$, and $comm = output(A_1)$.

assigned roles: classification result:

alltf:	*tagger*	*silent*	*undecided*	none
tagger	69,997	0	0	2,954

alltc:	*tagger*	*silent*	*undecided*	none
tagger	766	0	0	0
tagger (hidden)	0	0	0	72,185

random:	*tagger*	*silent*	*undecided*	none
tagger	22,149	0	0	1,541
silent	0	21,966	0	1,571
tagger (hidden)	0	0	0	12,780
silent (hidden)	0	0	0	12,944

random+noise:	*tagger*	*silent*	*undecided*	none
tagger	21,625	0	1	2,064
silent	53	3,679	17,687	2,118
tagger (hidden)	0	12	2	12,766
silent (hidden)	1	9	3	12,931

random-p:	*tagger*	*silent*	*undecided*	none
tagger	6,750	0	0	5,876
silent	0	13,445	0	11,983
selective	2,351	3,538	837	5,984
tagger (hidden)	0	0	0	5,562
silent (hidden)	0	0	0	11,082
selective (hidden)	0	0	0	5,543

random-pp:	*tagger*	*silent*	*undecided*	none
tagger	2,163	0	0	10,463
silent	0	4,429	0	20,999
selective	1,301	713	134	10,562
tagger (hidden)	0	0	0	5,562
silent (hidden)	0	0	0	11,082
selective (hidden)	0	0	0	5,543

Table 5: Assigned roles vs. classification results: Confusion Matrices for *tagging* behavior per scenario.

assigned roles: classification result:

alltf:	*forward*	*cleaner*	*undecided*	none
forward	10,427	0	0	2,104
forward (leaf)	0	0	0	60,420

alltc:	*forward*	*cleaner*	*undecided*	none
cleaner	0	578	0	124
cleaner (hidden)	0	0	0	11,829
cleaner (leaf)	0	0	0	60,420
random:	*forward*	*cleaner*	*undecided*	none
forward	2,400	0	0	1,091
cleaner	0	2,433	0	1,106
forward (hidden)	0	0	0	2,750
cleaner (hidden)	0	0	0	2,750
forward (leaf)	0	0	0	30,335
cleaner (leaf)	0	0	0	30,085

random+noise:	*forward*	*cleaner*	*undecided*	none
forward	2,294	0	63	1,134
cleaner	1	738	1,647	1,153
forward (hidden)	0	2	2	2,746
cleaner (hidden)	0	1	0	2,750
forward (leaf)	0	0	0	30,335
cleaner (leaf)	0	0	0	30,085

random-p:	*forward*	*cleaner*	*undecided*	none
forward	925	75	266	1,666
cleaner	0	1,355	0	1,663
forward (hidden)	0	52	0	3,265
cleaner (hidden)	0	47	0	3,217
forward (leaf)	0	0	0	30,480
cleaner (leaf)	0	0	0	29,940

random-pp:	*forward*	*cleaner*	*undecided*	none
forward	221	72	279	2,341
cleaner	0	610	0	2,386
forward (hidden)	0	14	0	3,322
cleaner (hidden)	0	19	0	3,267
forward (leaf)	0	0	0	30,480
cleaner (leaf)	0	0	0	29,940

Table 6: Assigned roles vs. classification results: Confusion Matrices for *forwarding* behavior per scenario.