In Search for an Appropriate Granularity to Model Routing Policies

Wolfgang Mühlbauer* Steve Uhlig♣ Bingjie Fu♣ Mickael Meulle□ Olaf Maennel♦
*TU Berlin/T-Labs, ♣Delft University of Technology, □France Telecom R&D, ♦University of Adelaide

ABSTRACT

Routing policies are typically partitioned into a few classes that capture the most common practices in use today [1]. Unfortunately, it is known that the reality of routing policies [2] and peering relationships is far more complex than those few classes [1,3]. We take the next step of searching for the appropriate granularity at which policies should be modeled. For this purpose, we study how and where to configure per-prefix policies in an AS-level model of the Internet, such that the selected paths in the model are consistent with those observed in BGP data from multiple vantage points.

By comparing business relationships with per-prefix filters, we investigate the role and limitations of business relationships as a model for policies. We observe that popular locations for filtering correspond to *valleys* where no path should be propagated according to inferred business relationships. This result reinforces the validity of the *valley-free* property used for business relationships inference. However, given the sometimes large path diversity ASs have, business relationships do not contain enough information to decide which path will be chosen as the best. To model how individual ASs choose their best paths, we introduce a new abstraction: *next-hop atoms*. Next-hop atoms capture the different sets of neighboring ASs an AS uses for its best routes. We show that a large fraction of next-hop atoms correspond to per-neighbor path choices. A non-negligible fraction of path choices, however, correspond to hot-potato routing and tie-breaking within the BGP decision process, very detailed aspects of Internet routing.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing Protocols*; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Internet (e.g., TCP/IP)*
General Terms: Algorithms, Experimentation, Measurement
Keywords: BGP, inter-domain routing, routing policies

1. INTRODUCTION

The Internet is composed of a large number of independently administered networks (Autonomous Systems or ASs), coupled by the inter-domain routing protocol (BGP) into a single globe spanning entity. Inter-domain routing is controlled by diverse policies, decided locally by each AS, and is not directly observable from available BGP data. Those policies act globally across the entire system [4]. Hence the topology of the inter-domain graph is not, in itself, enough to model the reality of inter-domain routing. Policies also need to be considered to capture the reality of the path choices made by each AS.

Policies are not easily defined [1] as they encompass the business and engineering decisions made by each AS, both commercial agreements (business relationships) and technical aspects (router configuration, inter-domain routing behavior, *etc.*). In this paper, we aim to capture the appropriate level of detail about policies to be used in a model of the Internet. Our ultimate objective, which is not achieved yet in this paper, is to build a model of the Internet with a sufficiently detailed view of the AS-level connectivity and its policies so as to be able to have useful predictive capabilities about BGP paths.

So far, models of the network structure have been mostly inter-domain level models that do not care about details of the ASs [5–7]. However ASs are not simple nodes in a graph. Rather they consist of routers spanning often large geographic regions. The internal structure of an AS *does* matter. It influences inter-domain routing, for instance via hot-potato routing [8,9]. Further, there are multiple connections between ASs, typically from different routers in different locations, which adds to the diversity of known routes [10, 11].

The main goal of this paper is to study the granularity of routing policies in the Internet as they are observed from BGP data from multiple vantage points. We do not blindly rely on existing notions of routing policies such as business relationship inference [5, 12, 13]. Rather, we rely purely on what we observe in BGP data and attempt to learn as much as possible about the "correct" level-of-detail needed to model actual routing policies. Our main concern is not to shrug off existing approaches, but to pinpoint their advantages and disadvantages and how they are related to one another.

Our approach is similar to that of [14], as we build an AS connectivity graph that enables the propagation of all routes present in observed BGP paths. To match observed routing, we introduce "agnostic" policies, since it is impossible to infer all of the details of an AS's policies without access to router configurations. However, we go beyond the agnosticism of [14] as we compare inferred policies with business relationships. The gained insights are important for our study of the right granularity to model routing policies.

Our work reveals two dimensions to policies: (i) which routes are allowed to propagate across inter-domain links (route filtering); and (ii) which routes among the most preferred ones are actually chosen (route choice) and thus observed by BGP monitors. In terms of the first dimension we show that the granularity of business relationships is largely consistent with observed paths. AS rela-

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

tionships provide the right abstraction to prevent unnecessary paths from propagating in a model of the Internet. For the second dimension, however, the classes of neighbors defined by business relationships are not precise enough. When only business relationships are used as policies in a model of the Internet, there are still many possible candidate paths among which the best path can be chosen. Business relationships are not sufficient to determine among all possible valid paths, which one should be chosen as best by the model to be consistent with observed BGP data.

To crystallize the choice of paths an AS makes, we introduce a new concept: *next-hop atoms*. Next-hop atoms capture the different sets of neighboring ASs that each AS chooses for its best routes. We show that a large fraction of next-hop atoms correspond to per-neighbor path choices. A non-negligible fraction of path choices, however, correspond to hot-potato routing and tie-breaking within the BGP decision process, very detailed aspects of Internet routing.

The remainder of this paper is structured as follows. Section 2 introduces the BGP data used and presents our AS-topology model. Section 3 analyzes the known bounds for policies studied in the literature. In Section 4, we search for the right granularity to model policies: we infer per-prefix filter rules (Section 4.1) and compare them with business relationships (Section 4.4). The insights gained are important for Section 5 where we discuss the difference between routing policies and path choices. In Section 5 we come up with a new abstraction that captures the selection of paths by ASs: next-hop atoms. The related work is described in Section 6. Finally, Section 7 concludes and discusses further work.

2. AS-TOPOLOGY MODEL

To study the granularity of policies, we need a topology model of the Internet. The measured routing data, used throughout this paper, is described in Section 2.1. Section 2.2 describes some properties of the AS connectivity observed in this data, which precedes our explanation in Section 2.3 of how the AS graph of our model is built from the observed paths.

2.1 Data

Different techniques exist to collect BGP feeds from an AS. One of the most common techniques is to rely on a dedicated workstation running a software router that peers with a BGP router inside the AS. We refer to each peering session from which we can gather BGP data as an *observation point*, and the AS to which we peer as the *observation AS*.

We use BGP data from more than $1,300$ BGP observation points, including those provided by RIPE NCC [15], Routeviews [16], GEANT [17], and Abilene [18]. The observation points are connected to more than 700 ASs, and in 30% of these ASs we have feeds from multiple locations.

As we are currently not interested in the dynamics of BGP we use a static view of the routes observed at a particular point in time. The table dumps provided by the route monitors are taken at slightly different times. We use the information provided in these dumps regarding when a route was learned to extract those routes that were valid table entries on Sun, Nov., 13, 2005, at 7:30am UTC and have not changed for at least one hour. In future work we are planning to also incorporate the AS-path information from BGP updates.

Our dataset contains routes with $4,730,222$ different AS-paths between $3,271,351$ different AS-pairs. An AS-level topology is derived from the AS-paths. If two ASs are next to each other on a path we assume they have an agreement to exchange data and are therefore neighbors in the AS-topology graph. We are able to identify $21,159$ ASs and $58,903$ AS-level edges.

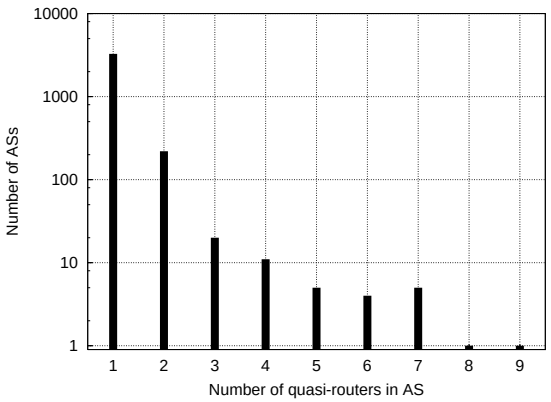

Figure 1: Number of quasi-routers per AS.

2.2 AS-Level Connectivity

As already shown in [7,14], for an AS topology model to capture route diversity, ASs cannot be considered atomic entities. In order to represent the intra-domain routing diversity, we allow each AS to consist of multiple quasi-routers. A *quasi-router* represents a group of routers within an AS, all making the same choice about their best routes. Thus the "quasi-router topology" does not represent the physical router topology of a network, rather the logical partitioning of its observed path choices. An AS has to be modeled with multiple quasi-routers if it receives and chooses as best multiple paths towards at least one prefix.

Figure 1 provides the number of quasi-routers per AS that are required to capture BGP path diversity. In any data analysis results of this paper, we do not consider stub ASs, i.e., ASs that appear as the last AS hop on any AS path in our data (pure originating AS)[1]. Among the $3,535$ remaining ASs, 267 require more than a single quasi-router. Only 2 ASs need as many as 8 and 9 quasi-routers to account for their observed routing diversity. Typically, well-known tier-1 ASs require several quasi-routers. This is consistent with [10] which showed, based on active measurements, that tier-1 ASs have high path diversity. On the other hand, a low number of quasi-routers per AS is due both to the sampling of the available paths of the observed BGP paths, as well as the loss of BGP routing diversity inside ASs [19].

Diversity of the AS paths is strongly related to the AS-level connectivity. Figure 2, in which we consider the same $3,535$ ASs as in Figure 1, shows a scatterplot of the relationship between the number of required quasi-routers and the number of neighboring ASs. We observe that ASs that do not have many neighbors also tend to have a small number of quasi-routers. Highly connected ASs on the other hand may have many quasi-routers, although this is not necessarily always true. Some ASs have hundreds of neighbors, yet a single quasi-router is enough to account for their routing diversity.

As previously stated, there are two reasons why an AS requires several quasi-routers: (i) the AS receives and selects as best multiple paths towards a given prefix from a given neighbor; and (ii) the AS receives and selects as best different paths towards a prefix but from different neighbors. From Figure 2, we can observe that highly connected ASs have a far larger number of neighbors than quasi-routers. ASs thus select a very small subset of best paths compared to the number of paths they may receive from their neighbors, for any prefix. Note that the first reason why an AS might need several quasi-routers does not seem to be common. For only

[1]Although being transit domains, some ASs may only have one AS neighbor after removing stub ASs.

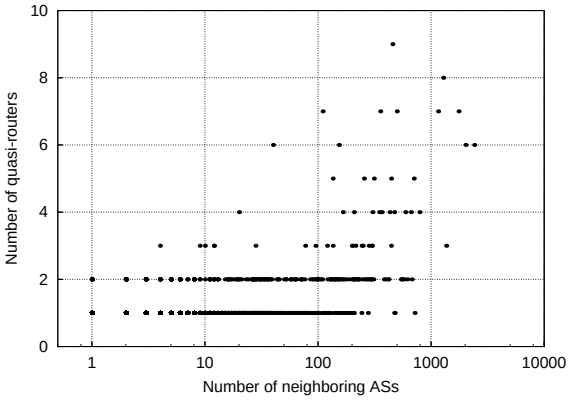

Figure 2: Relationship between number of neighboring ASs and number of quasi-routers.

623 pairs of neighboring ASs do we observe in the data that an AS chooses from one of its neighbors more than one path towards at least one prefix. Further, in only 19 cases do we observe an AS receiving more than 2 distinct paths from a given neighbor towards at least one prefix.

2.3 Building a Quasi-Router-Level Graph

For our study of the granularity of routing policies we need a topology model of the Internet. We capture the inter-domain connectivity via an AS-topology graph as extracted from the BGP data (see Section 2.1). In order to represent the intra-domain routing diversity, we allow each AS to consist of multiple quasi-routers.

To ensure the connectivity of our model is minimal, the topology is built when assigning to quasi-routers AS path suffixes observed in the data. A suffix s of an AS path P is any substring Q such that $P = Qs$. The AS topology we create has as few quasi-routers per AS as possible, and an edge exists between two quasi-routers if some suffix has to be propagated between the two quasi-routers.

Our assignment works on a per-prefix basis. First we set all quasi-routers as free to be assigned paths towards the considered prefix, and set all suffixes towards this prefix as to be assigned. Then, as long as there are suffixes that are not assigned, we try to assign them by starting with those suffixes closest to the originating AS(s) of the prefix. When assigning suffixes, we first re-use existing connectivity between quasi-routers. If no link between the first two ASs on the suffix can be re-used for this prefix, we then create a new link between a free quasi-router in the first AS on the suffix and the next AS. Note that the creation of the topology (links between quasi-routers) follows directly from the path assignment. Due to space limitations, we do not explain in detail how our AS topology is built.

The number of necessary quasi-routers in an AS is not the only parameter that matters for allowing an AS topology model to reproduce the paths observed in BGP data. Even though only few quasi-routers might be necessary to account for the routing diversity of an AS [14], the way quasi-routers between two ASs are connected also matters. If, in general, an AS requires the same number of inter-domain links as it has neighboring ASs, it means that even though this AS might have many neighbors, only a single neighbor at a time is used as next hop AS in the best routes for any prefix. If an AS in our model has substantially more inter-domain links than neighboring ASs on the other hand, it means that the considered AS uses several neighboring ASs for its best routes towards some prefixes.

3,150 among the 3,535 transit ASs of our data require a single inter-domain link with any of their neighboring ASs. Only 386 ASs require more than one inter-domain link per neighbor, and 41 ASs more than 2 inter-domain links. As seen from BGP data, only a very small fraction of the ASs choose their best paths from several neighbors at the same time towards any of their prefixes.

3. BOUNDS ON POLICY GRANULARITY

To find an appropriate way to model policies in the Internet, it is important to start with realistic bounds that define the finest and coarsest granularities at which policies are applied in the Internet. There are two ends to this spectrum. The finest granularity is the one of BGP atoms [20, 21], which are sets of prefixes originated by a given AS that receive equivalent treatment by routers in the Internet. BGP atoms are as fine as the set of policies that the observed BGP paths encounter, which can be as fine as on a per-prefix basis. The coarsest granularity does not depend on the originated prefixes, but only on the neighbors from which routes are received. It is the granularity of business relationships. ASs may configure policies as coarse as per-neighboring AS, hence treating all prefixes, received from a given neighbor, in the same way.

3.1 BGP Atoms

For inter-domain routing, each prefix is handled independently from other prefixes. However, groups of prefixes may receive equal treatment by a given set of BGP routers, due to the granularity of routing policies. The analysis of BGP routing tables has shown that clusters of prefixes originated by given ASs undergo the same routing policies [20, 21]. Groups of prefixes (originated by a given AS) that receive *equivalent treatment* by a set of BGP routers are called *BGP atoms* [20, 21]. As BGP monitors see only a sample of the outcome of routing policies through observed AS paths, not the policies themselves, a *BGP atom* is defined as a set of prefixes that share the same set of AS paths as seen from a set of BGP routers [21]. Two prefixes are put in the same BGP atom if their AS-PATH is the same, as seen by all observation points. The finest granularity of a BGP atom is a single prefix, whereas the coarsest is all the prefixes originated by an AS (or a set of origin ASs in the case of MOAS prefixes [22]).

Atoms' sizes vary across and within origin ASs. A large fraction of atoms consist of a single prefix, while some atoms consist of tens of prefixes. As BGP atoms are defined with respect to a given set of vantage points, policies applied by single ASs might be coarser than per BGP atom.

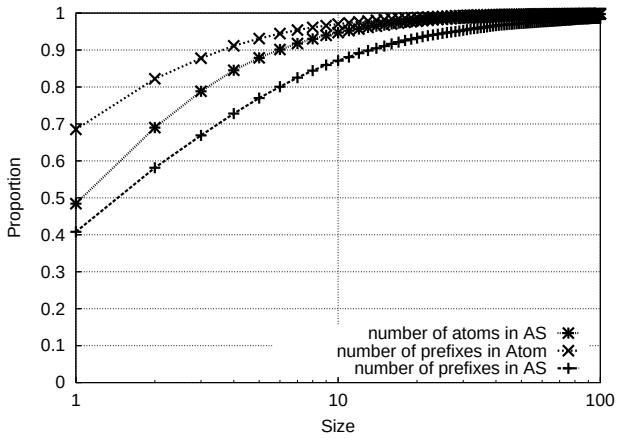

Figure 3: Atoms structure for dataset.

To compute atoms, we use an approach similar to [21]. We put two prefixes $p1$ and $p2$ in the same BGP atom if their AS-PATH is the same, as seen by all our observation points. Figure 3 presents the atoms structure of the dataset. The graph displays three cumulative curves: the number of prefixes per origin AS, the number of prefixes in each atom, and the number of atoms in each AS. We observe that the distribution of the number of prefixes per origin AS is virtually the same as in [20] which relies only on RIPE NCC data. More than 40% of the origin ASs advertise a single prefix: at least 40% of the atoms hence consist of a single prefix. The curve giving the number of prefixes per atom shows that 70% of the atoms consist of a single prefix. The RIPE data used by [20] had 60% of single prefix atoms.

We believe that we observe finer atoms in the data compared to [20] because our data provides a more extensive coverage of the actual BGP paths. With an increasing number of paths, we also observe the effect of more policies, leading to smaller atoms due to more diverse path choices. The curve in Figure 3, showing the number of atoms in each AS, confirms that data used in this study sees ASs that have more atoms than that observed using RIPE. In our data, about 30% of the ASs contain two or more atoms, whereas [20] observed only slightly more than 25% of ASs with two atoms or more.

3.2 Business Relationships

Business relationships rely on the coarsest granularity possible for policies: filtering rules defined on a per-neighbor basis. More details on business relationships and their inference can be found in Appendix A.

As business relationships are the most popular model for policies in the literature, we simulate the path choices in our model, when the only policies configured are inferred business relationships. Then, we compare the paths chosen in the model with those observed in BGP data. Customer-provider and peering relationships are inferred by applying the CSP algorithm [23] to the data (see Appendix A).

We rely on C-BGP [24, 25] to compute the outcome of the BGP decision process and the set of learned routes at every router of our AS-level topology. C-BGP computes the steady-state choice of the BGP routers after the exchange of the BGP messages has converged.[2] This allows us to perform large-scale simulations for single prefixes on topologies with more than $21,547$ routers split among $21,169$ ASs in approximately 2 minutes with 300 MB of memory consumption. Each quasi-router in our model corresponds to a router in the C-BGP simulation.

AS-Paths which agree	14.5%
AS-Paths which disagree	85.5%
due to	
route not available	60.9%
route learned but not selected	24.6%

Table 1: Agreement between observed and simulated routes when business relationships are used as policies.

Table 1 shows the consistency between the path choices simulated in the model with business relationships configured as policies, and the paths observed in the data. For each observed path, we check if there is at least one quasi-router that selects the observed AS path as best route in the simulation.

[2]We choose to assign IP addresses such that the high order 16 bits are set to the AS number and the low order bits are a unique ID for the quasi-router. In case of a tie-break a quasi-router prefers AS paths announced by quasi-routers with low IP addresses.

Only 10.1% the paths agree between the simulation and the observations. For 60.9% of the paths, the corresponding path is not even propagated to the AS that should observe that path in the simulations. Only 24.6% of the paths are learned by the right AS but not selected as best path by any quasi-router of that AS.

We find these results disappointing. Introducing business relationships does not seem to solve any inconsistencies between the paths propagated in our model and the routes actually observed in the Internet.

3.3 Atoms vs. Relationships

We believe that neither BGP atoms nor business relationships give an ultimate answer to the problem of which granularity should be used for modeling routing policies.

On the one hand, business relationships appear too coarse, as they result in high inconsistencies between the paths propagated in our model and the routes actually observed in the Internet. We want to point out that this does not necessarily mean that business relationships are "wrong". However, it is unclear to what extent having per-neighbor policies is responsible for this high inconsistency.

BGP atoms, on the other hand, also have shortcomings. Two prefixes are put in the same atom if their AS-PATH is the same, as seen by *all* our observation points. According to this definition BGP atoms describe policies *across many* ASs, i.e., observation points. We believe that relying on BGP atoms is therefore dangerous for our study, as atoms do not discriminate different inter-domain links and parts of the topology. For example, BGP atoms do not capture situations where a large fraction of the policies in the Internet are defined as coarse as per-neighboring AS, while only a small subset of ASs configure policies on a per-prefix level. In this case, BGP atoms are prone to generalize and would suggest that probably all ASs have their policies defined on a per-prefix level.

4. IN SEARCH FOR THE RIGHT GRANULARITY

Given the results of Section 3, one may wonder whether business relationships or BGP atoms are the right way to model policies in the Internet. Therefore, we now start our search for the appropriate granularity of policies from scratch and rely on the finest granularity possible: per-prefix filtering. We identify fine-grained policies by analyzing what we see in our data of Section 2.1 and comparing it to the routes selected by our model without implemented policies. The motivation behind this approach is to compare the obtained per-prefix filters with coarse-grained policies as imposed by inferred business relationships. The gained insights will be important in Section 5 when we propose a new abstraction for routing policies.

As it is impossible to extract information about the implementation of policies only based on observed BGP data, we restrict ourselves to per-prefix filtering: if there is a disagreement between some observed path and the corresponding route selected in our model, a set of filter rules is identified to prevent the propagation of "wrong" paths. Section 4.1 explains in detail how sets of filter policies are computed. Given that we restrict ourselves to filtering policies, how much freedom do we have in placing those filters? Filtering between different AS-level peerings may have the same effect in terms of path propagation for the observed path. Section 4.2 tries to estimate the amount of freedom we have in terms of equivalent policies when trying to achieve consistency between best routes in our model and observed AS paths. In Subsection 4.3, we make an important step in our search for the appropriate granularity of policies. Given a large set of per-prefix filtering rules

that has been computed exclusively based on observed data, we try to find out if there is possible aggregation across prefixes. More precisely, we check whether there are locations on the AS connectivity graph that seem to benefit more frequently from a filter than others. As we detect some very popular locations for filtering in Section 4.3, we conclude that the implementation of actual routing policies is somewhere in-between per-prefix and per-neighbor policies. Therefore, we compare in Section 4.4 inferred business relationships with the per-prefix filters we detected using the approach of Section 4.1.

4.1 Inferring Filters

We now we describe how to identify sets of per-prefix policies in order to obtain agreement between the routes selected in our model and those observed in the data. The guideline in this approach is to rely only on what we see in the data. We account for this basic principle as follows: First, the physical connectivity of our AS topology of Section 2.3 is sufficient to make the propagation of all observed AS paths possible, if policies are to be installed properly. Second, policies are introduced on a per-prefix basis, the finest granularity for which policies can be configured. Third, we want to make as weak assumptions as possible about where to place a policy. If an observed AS path is not selected in the topology model of Section 2.3, we have a large choice about where and what policy to introduce. Different policy types and different AS-level peerings may have the same effect in terms of path propagation for the observed AS path. Therefore, we try to identify multiple "candidate" policies first and in a later step (cf. Section 4.4) we will use heuristics to pinpoint likely policies.

The example in Figure 4 illustrates the many possible locations for policies, if the only goal is to allow for the propagation of an observed AS path. We observe at AS 7 an AS path `7-6-5-4-3` originated by AS 3. However, reproducing the BGP route selection in this topology without any policies will show that AS 7 selects the shorter path `7-2-3` to reach the prefix. In this case, a preference policy at AS 7 or filtering at least one link both of the paths `7-2-3` and `7-1-2-3` will have the same effect in terms of the propagation of the observed AS path `7-6-5-4-3`. Note that it is even possible to apply an arbitrary subset of all "candidate" policies that will have the same effect.

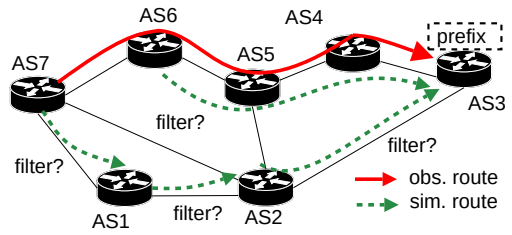

Figure 4: Filtering - Example.

In order to reproduce BGP route selection in our AS-level topology, we again use C-BGP [24, 25]. As a consequence, we know for every router in our model which routes it learns to reach a prefix and also which route is selected as best. According to the assignment of observed AS path suffixes to quasi-routers in our AS-topology graph (see Section 2.3), many of the routers in our model are supposed to select a specific path to reach a certain prefix. However, without properly configured policies, the paths chosen by our model might not be the same as those observed in the data. A *mismatch* is referred to a situation where a quasi-router chooses an AS path which is inconsistent with the path assignment of Section 2.3. In our approach, each mismatch gives a hint about where policies are required. We now distinguish between two different cases of mismatch.

The first case of mismatch can occur when, a router does not select the path consistent with the assignment of Section 2.3, due to the existence of some shorter AS paths. In this case, we will introduce per-prefix BGP *filters* on the link from the announcing neighbors to prevent the shorter paths from being propagated to the router. In Figure 4, both AS 1 and AS 2 will propagate routes to AS 7 which are shorter than the observed AS path `7-6-5- 4-3`. In the following, we denote a filtering rule in our model between AS X and AS Y, where Y does not propagate a prefix towards X, by $X \nleftarrow Y$. Thus, configuring the filter rules $7 \nleftarrow 1$ on link `7-1` and $7 \nleftarrow 2$ on link `7-2` can be used to obtain the observed path at AS 7.

The second case will occur if a router does not select the "correct" AS path due to a wrong "tie-breaking" decision in our model. Provided some router receives multiple routes with equal AS path lengths, the BGP decision process will have to break ties, e.g., by preferring the route learned from the neighbor with the lowest IP address. We ignore those situations since no policy is identified. Indeed, we cannot be sure whether a policy is actually needed to get the correct propagation. We do not want the uncertainty of the BGP decision process and its implementation to impact our study of the granularity of policies. Reconsidering the example in Figure 4, we see that AS 5 may not select the observed suffix `5-4-3` due to a "wrong" tie-breaking decision: the C-BGP simulation will prefer the path `5-2-3` if the router of AS 2 has a lower IP address than the router of AS 4.

Let us now define three notions that will be used to explain the detection of filtering policies:

Candidate filter: A per-prefix filter rule which helps to allow the selection of an observed path as best route in our model. In general, several candidate filters (e.g., a filtering combination) will be needed. Additionally, shorter paths do not necessarily have to be filtered at the location of the mismatch. To obtain the observed path at AS 7 in Figure 4, filtering on the link `2-3` has the same effect as having filters on both `7-1` and `7-2`. Altogether, we identify four candidate filters in our example: $7 \nleftarrow 2$, $7 \nleftarrow 1$, $1 \nleftarrow 2$, $2 \nleftarrow 3$.

Filtering combination: A set of filter rules for a mismatch which satisfies two conditions: (i) Applying *all* filters in this set clears the mismatch, i.e., there will be agreement between the observed suffix path assigned to a quasi-router, and the route currently selected in our model; and (ii) the set of policies in this set is *minimal*, i.e., if *any* policy from a filtering combination is removed the mismatch will not disappear. In the example in Figure 4, there are three filtering combinations:

(1) $7 \nleftarrow 2$ and $7 \nleftarrow 1$
(2) $7 \nleftarrow 2$ and $1 \nleftarrow 2$
(3) $2 \nleftarrow 3$.

However, the set of filter rules $7 \nleftarrow 2$, $7 \nleftarrow 1$ and $1 \nleftarrow 2$ is not considered as a filtering combination, as either $7 \nleftarrow 1$ or $1 \nleftarrow 2$ can be removed while the router of AS 7 still chooses the assigned suffix `7-6-5-4-3`.

Dependency graph: A data structure used to store the identified filter candidates and their dependencies for a certain prefix. Nodes in this graph represent candidate filters or mismatches whereas directed edges between nodes reflect dependencies. The direction of the edges is determined by our algorithm. Basically, the algorithm recursively walks back from the "mismatched AS" to the originating AS, detecting filters along the way. Dependency edges are always directed

towards filters which are closer to the originating AS. The idea now is that a "filter node" is not needed provided that (i) *all* its children nodes or (ii) *all* its parent nodes in the dependency graph are used or (iii) if there are no parent and children respectively. Figure 5 shows the dependency graph for the mismatch at AS 7 in Figure 4. There are five nodes, with one representing the mismatch at AS 7 and the remaining nodes the four candidate filters. Assume that filter 2↚3 is used. In this case, 7 ↚ 2 as well as 7 ↚ 1 are redundant, with all their children nodes (filter 2 ↚ 3) already used.

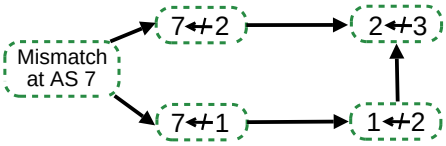

Figure 5: Dependency graph for mismatch in Figure 4.

The benefits of this data structure are two-fold. First, compared to keeping all filtering combinations, the dependency graph scales as its size is bounded by the number of links and ASs in our topology. Second, it prevents losing information about possible dependencies between detected filters rules.

Our algorithm to compute a set of candidate filters for a given prefix is summarized in Figure 6. It takes as input the observed routes to a specific prefix, an AS topology including the assignment of observed AS paths to quasi-routers (see Section 2.3) and the routes selected in our model when simulating BGP route propagation with C-BGP. For each mismatch, a set of candidate filters is identified and inserted into the dependency graph. The result is a dependency graph for the prefix, with all candidate filters being associated to at least one mismatch.

```
foreach observed path p to the given prefix
    start at originating AS and walk to observation AS
    foreach hop h of path p
        o = suffix of p from AS h to observation AS
        s = simulated path at the router assigned for suffix o
        o_length = length of suffix o
        if s not equal o
            add "mismatch" m to dependency graph
            findCandidates(h, o_length, m, 1)

sub findCandidates(hop h, length l, from_policy f_from, depth r)
    if (r > threshold) or (h == originating AS) [1]
        terminate
    foreach physical neighbor n of h [2]
        n_length = length of path announced from n
        if n_length < l
            add "FILTER " f_new to dependency graph
            findCandidates(n, l − 1, f_new, r + 1)
        add link from f_from to f_new in dependency graph
```

[1] not all termination criteria listed.
[2] some neighbors can be skipped, not shown here.

Figure 6: Computing candidate policies for a prefix.

As shown in Figure 6, the algorithm proceeds by consecutively looking at all routes observed for the prefix. For each route, we walk along the AS path from the originating AS to the observation AS and check at each hop for an inconsistency, i.e., a disagreement between the observed suffix path and the simulated route at the assigned quasi-router. If there is a mismatch, the function *findCandidates* is called recursively to identify filter candidates.

The recursion serves the purpose of considering filters that are not directly located at the mismatch but closer to the originating AS. The basic idea is that the function *findCandidates* has as a parameter the current AS hop h and recursively calls itself on neighboring ASs from which it learns routes which are too short in terms of AS path length.

To know which routes need to be filtered, we use another parameter l, the maximum path length which an AS is allowed to propagate. Provided that AS h of *findCandidates* selects in the simulation a route with a strictly shorter AS path than l, a filter between h and AS c – the AS from which this recursion has been called – will be added to the list of candidate filters. At the same time, we insert a dependency edge between the new filter and the candidate filter d_n found at AS c.

In general, recursion terminates when we arrive at an originating AS or when the current AS does not select a route shorter than the maximum allowed path length l. There are many other situations where recursion is stopped. For example, we allow the specification of a threshold for the maximum recursion depth. Additionally, no recursion is required if we arrive at an already visited AS hop. In our topology of Figure 4, the filters 7↚1 and 7↚2 are detected while looking at neighbor AS 1 and AS 2 at recursion depth 1. While at AS 1, there will be a recursive call for AS 2 with recursion depth 2. However, AS 2 has already been visited and thus the candidate filters have been already computed. Recursion can thus be stopped safely without losing information.

4.2 Freedom in Filters Location

We now apply the algorithm in Figure 6 to compute *candidate combinations* on the AS-topology of Section 2.3. The goal is to give an estimate of the choice we have in terms of filter candidates when trying to achieve consistency between best routes in our model and observed AS paths. For this, we randomly select an extensive number of prefixes, called `psample` (see Table 2).

# prefixes	50,000
# originating ASs	10,575
# distinct AS paths	2,267,296
# prefixes per AS path	
- mean	3.6
- standard deviation	11.8
# distinct AS paths to a prefix	
- mean	161
- standard deviation	42
# mismatches per prefix	
- mean	3,328
- standard deviation	5,191

Table 2: Statistics on `psample`.

`psample` contains more than 2 million AS paths to 50,000 prefixes. For each prefix, we have a mean of 160 distinct AS paths, with an average of 3.6 prefixes sharing a common AS path. While running our algorithm, we detected in total more than 10 million mismatches, i.e., AS hops that do not select the "correct " suffix of an observed route. Even for a single prefix, the number of detected mismatches is considerable, with 3,328 on average.

To study the impact of recursion on the number of filter candidates found, we run our algorithm with three different thresholds for the maximum recursion depth. The results are summarized in Table 3. Allowing filters only on links incident to the AS hop with the mismatch (recursion depth 1) results in an average of 32.9 filter candidates per mismatch. This number is surprisingly high but can be explained by some ASs having a large number of neighbors from which routes have to be filtered. With a recursion depth of 2 (3), this increases to more than 1,000 (3,000) candidate filters on average.

	recursion depth 1	recursion depth 2	recursion depth 3 [1]
mean	32.9	1,103	2,952
standard deviation	116	4,518	12174
min	1.0	1.0	1.0
max	1,847	49,040	80,050

[1] only for a subset of 2,000 prefixes

Table 3: Number of candidate filters per mismatch for `psample`.

To measure the freedom we have in combining those candidate filters, we use the notion of *filtering combinations* defined in Section 4.1. We slightly modify the recursive function *findCandidates* of our algorithm in Figure 6 to return the number of possible filtering combinations. Recall that each filtering combination ensures that no path is selected at the current AS hop which is strictly shorter than the maximum allowed path length. In general, there are multiple "bad" neighbors from which we have to filter out shorter paths. The number of possible filtering combinations is the number of non-empty subsets of lines from the filtering combinations that contain the "bad" neighbor.

Obviously, we only obtain a single filtering combination when recursion is terminated at depth 1. However, with a maximum recursion depth of 2 the average number of filtering combinations per mismatch is already in the order of 10^{500}, increasing to $10^{13,000}$ for a maximum recursion depth of 3. Note that these numbers are only rough estimates. Still, they illustrate the freedom we have in filter locations. There would be even more choice if we did not restrict ourselves to non-redundant *filtering combinations* and were to allow other policies, e.g., local-preference.

4.3 Popularity of Filters

In the previous section, we computed an extensive number of candidate filters. Applying those per-prefix filters is supposed to ensure the propagation in our model of observed paths. The main idea now is to check whether there are filter locations that are more popular than others. We call a filter on an AS-level link *popular* if the link is identified as a possible filtering location for many prefixes by our algorithm of Section 4.1. A large number of such popular filters suggests that per-prefix policies are too fine and should be aggregated into coarser policy entities.

To detect popular filters, we run the algorithm of Section 4.1 on the observed routes of `psample` (see Table 4). Using a maximum recursion depth of 1 and 2 in our algorithm reveals the impact of the recursion depth on the popularity of the identified filters. For each directed AS-level link, we count the number of prefixes for which a filter candidate is identified as "useful" on that link. The distribution of filters popularity for both recursion depths is plotted in Figure 7.

Figure 7 shows that some filters are more popular than others. While for a recursion depth of 1, less than 5% of the detected candidate filters are useful for at least 10,000 prefixes (out of 50,000), this is more than 30% for recursion depth 2. A similar trend is observed for larger recursion depths. The reason for this may be that large recursion depths add a lot of noise, i.e., they identify candidate filters at locations which are unlikely to be related to the mismatches we try to fix.

Table 4 provides further details about the popularity of filters. There are some locations for filtering which seem to be very popular. With a maximum recursion depth of 1, 5% of the identified filter candidates are "useful" for more than 8,000 prefixes.

At the same time, we see filter candidates that are identified for only a very small number of prefixes. 25% of the detected filter

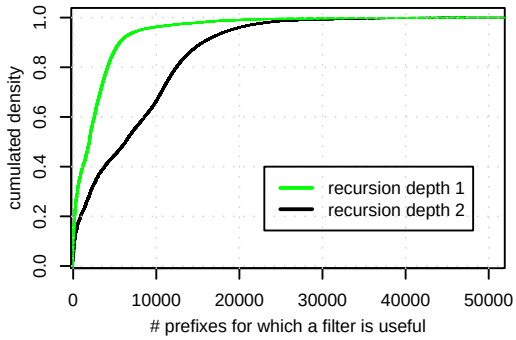

Figure 7: Popularity of filters in `psample`.

Percentile	25%	50%	75%	90%	95%	100%
#prefixes (depth 1)	236	1,888	3,604	5,548	8,004	46,921
#prefixes (depth 2)	1,480	6,237	11,389	15,523	18,896	47,032

Table 4: Popularity of filters for recursion depth 1 and 2 in `psample`.

candidates affect less than 236 prefixes (out of a total of 50,000) if a recursion depth of 1 is used. For a recursion depth of 2, this number increases to 1,480. Selecting the 2,290 most popular filtering locations for recursion depth 1 (*pfilters*), we check how many of the filter candidates for recursion depth 2 would be redundant with them, i.e., would not be needed to achieve agreement between observations and the routes in our model if the filters in *pfilters* were configured. For this purpose, we take the computed dependency graph of recursion depth 2 and initially mark each filter in *pfilters* as "covered". Then, other filters in the dependency graph can be recursively marked as "covered" if either all children policies or all parent policies are already marked. By doing so, we see that the average ratio of covered filters is 75%. This number is surprisingly high given that there are many prefixes with more than 60,000 filters being detected for a maximum recursion depth of 2.

The main lesson of this section is that a non-negligible part of our filter candidates can be aggregated into coarser policy entities if the only goal is consistency between propagation in our model and the observed data. Higher recursion depths are not very helpful. They add more noise thereby making it more difficult to identify popular locations for filtering.

4.4 Revisiting Business Relationships

With regards to the correct granularity to model inter-domain routing policies, neither business relationships nor BGP atoms appear to be the ultimate solution (see Section 3.3). In this section, we revisit business relationships and try to gain more insight into their shortcomings by comparing them with popular filters (see Section 4.3). The lessons we learn will be important when we develop a new abstraction for routing policies in Section 5.

Business relationships have two consequences for route propagation and selection: *preference* of certain routes and *no-transit* for some routes. First, network administrators may favor longer AS paths over shorter ones due to economic reasons. In general, routes learned from customers will be preferred over routes announced over peering links and peering routes will be favored over provider routes. Second, multi-homed stub ASs want to avoid being used as transit. For this reason, routes learned from provider and peering neighbors are not propagated to other provider or peering ASs.

We first try to shed light on the impact of the *no-transit* principle on route propagation. In Section 4.3 we identified filter candidates and found that there are some popular locations for filtering. The idea is now to compare such filter candidates with business relationships and to find out whether some of the popular filters in our per-prefix approach possibly implement *no-transit* policies in the AS-relationship "world". If so, this can be seen as a reason to consider business relationships as some form of routing policy.

We compare business relationships with our candidate filters as follows: as a first step, popular locations for filtering are identified. According to Table 4, 5% of the filters found are useful for more than $8,000$ prefixes. We select those filters and obtain a total of $2,290$ popular filters (see Table 5).

Based on our AS-level topology, we then compute all ASs triples. Altogether, there are more than 30 million such triples in our topology. The next step is to identify triples which violate the so-called *valley-free* property. In our terminology, a *valley* is a triple `A-B-C` along which no route should be propagated if the *no-transit* rule is correctly enforced. Let us assume that AS C and AS A are both providers of AS B according to the inferred business relationships. In this case, AS B will not announce any route learned from one of its providers to the other provider. According to Table 5, we find more than 5 million valleys.

total # of edges (directed)	117,822
total # of triples	30,351,164
total # of valleys	5,383,862
total # of (popular) filters	2,290
# filtered triples	991,268
# filtered valleys	602,619
ratio: filtered valleys to filtered triples	60.7%
# filters in at least one valley	2,283

Table 5: Business relationships vs. popular filters.

Now we check how popular filters and valleys are related to each other. For this purpose, we collect all triples `A-B-C` such that any popular filter appears as either `A-B` or `B-C`. This results in $991,268$ filtered triples. Surprisingly, 60.7% of the filtered triples are valleys according to our inferred business relationships. At the same time, almost all popular filters ($2,283$) are applied on AS-level links which are part of valleys. Popular filters hence frequently correspond to a non-transit policy, a situation where according to business relationships no path should be propagated. Henceforth, we conclude that the popular filters we identified suggest that the valley-free property used to infer business relationships is indeed correct.

However, the question remains of why using inferred business relationships exhibits this high level of disagreement when comparing the routes selected in our model with those observed in the data. As mentioned above, business relationships impact route propagation in two ways: *no-transit* and *preference*. Given our results, we believe that an insufficient or incorrect implementation of the *no-transit* principle is not the actual reason for these inconsistencies. Therefore, we now study the effectiveness of business relationships in preferring the "correct" observed path.

For this purpose, we again take the AS-topology of Section 2.3 and use business relationships inferred with the CSP algorithm [23]. Then, we run a simulation with C-BGP to compute the selected routes for every router to each prefix. The goal is to find out how much choice each router has to select a best path. In spite of business relationships, a router may still have the choice between a set of equally preferred routes. Therefore, we determine for each observed path whether the observing AS learns it from a provider, peer or customer AS according to the inferred AS relationships. Then, we look at the corresponding AS and quasi-router in our simulation and count the number of learned path which are of the same "type" as the observed path, i.e., also a customer, provider or peering path. Figure 8 shows the distribution of this number of alternative path over all observed paths.

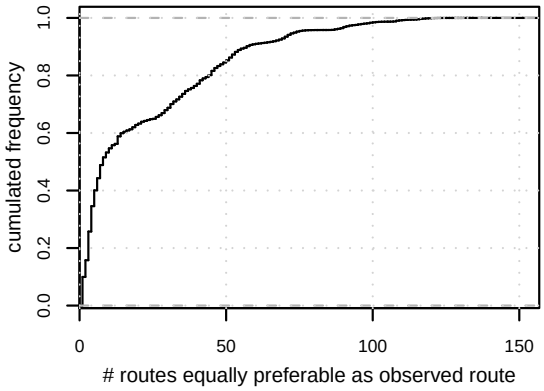

Figure 8: Business relationships: freedom in path choice for observed paths at observation points.

For only approximately 10% of the observed paths, there is a single path from which to select the best one. However, for more than 10% of all observed paths, we obtain more than 50 paths that are equally as good as the observed path, i.e. belong to the same type of path (customer, provider, peering). Given these results, we believe that business relationships do not reveal sufficient information about the actual *preference* policies used in the Internet. They only tell us that customer routes are preferred over peering routes, and peering routes over provider routes. Still, an AS may learn multiple customer, provider or peering routes for the same prefix. In such a case, business relationships cannot tell which one of the equally good routes should be selected as best.

To conclude this section, we state that the main problem inherent to relationship inference is the incomplete information it provides about the actual preference of paths.

5. FROM ROUTING POLICIES TO PATH CHOICES

Section 4 showed that modeling policies both as per-prefix filters and as business relationships has severe drawbacks. On the one hand, relying on business relationships is more scalable as less configuration is required in the model. Unfortunately, inferred relationships are not enough to lead to correct path choices. Per-prefix filtering, on the other hand, allows for models highly consistent with observed path choices, but it is not scalable as its granularity is the finest possible. If we now want to answer the question of what is the right granularity to implement routing policies in an Internet-wide model, we realize that do not have a definitive answer. Our conclusion so far is that business relationships are, in general, the right way to set routing policies in a model. However, predicting path choices requires more details about routing policies: one also has to guess which path to select as best from a set of equivalent paths, all permitted by policies.

To make the discussion more concrete, we need to introduce some concept that will crystallize this choice of the paths some AS performs. We call it the *next-hop atom*. A *next-hop atom NH* of an AS X is a subset of X's neighbors that X chooses as next-hops for

its best routes towards a given set of BGP atoms[3]. All BGP atoms for which we see that an AS uses the same set of neighbors for its best routes belong to the same next-hop atom. The aim of *next-hop atoms* is to capture the distinct sets of neighboring ASs an AS requires to describe its path choices towards groups of prefixes. Note that next-hop atoms do not reveal why some AS prefers some paths to others. Next-hop atoms only describe the choice ASs make, not the reasons for their choice.

Figure 9 illustrates an example of the choice of paths made by AS *X* towards five different BGP atoms. AS X is composed of two quasi-routers, QR_X1 and QR_X2. It has three neighboring ASs: A, B and C, each composed of a single quasi-router. The best path, AS X chooses towards BGP atom 1, has as next-hop AS A. To reach atoms 2 and 3, X uses as its next hop AS B, whereas the best paths towards both atom 4 and 5 go through AS B and C. In this example, AS X requires two quasi-routers because it has to choose two different best paths towards atoms 4 and 5.

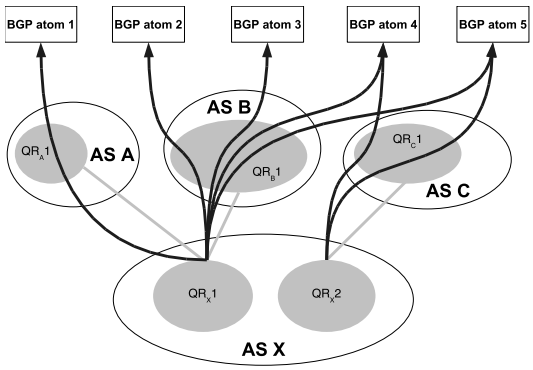

Figure 9: Example of path choices and next-hop atoms.

In the case of the example in Figure 9, AS *X* has three different next-hop atoms: NH_1 contains next-hop A towards BGP atom 1, NH_2 contains next-hop B towards BGP atom 2 and 3 (since AS X chooses its best routes towards BGP atom 2 and 3 via AS B), and NH_3 contains next-hops B and C towards BGP atom 4 and 5 (because AS X chooses its best routes towards BGP atom 4 and 5 via AS B and AS C). Among all possible combinations of next-hop ASs, only a subset will actually be used to send traffic towards BGP atoms. In our example, we only need three distinct combinations of neighboring ASs towards the five considered BGP atoms. A next-hop atom captures the coarsest granularity (across prefixes) at which an AS chooses its best paths in distinct ways (among its neighbors).

The reason to define next-hop atoms in terms of BGP atoms is that BGP atoms define the finest granularity at which sets of prefixes share the same path choices. One might choose to use prefixes instead of BGP atoms.

Now that we have the concept of next-hop atoms to capture the granularity at which ASs select their paths, we can study the observed granularity at which ASs choose their paths. The simplest way an AS can select its best paths is by always using the same set of neighbors for all prefixes. Such an AS would have the same next-hop atom towards all prefixes. Single-homed ASs are in this situation as they have a single neighbor from which to choose their paths. Large transit providers on the other hand are expected to have a large number of different next-hop atoms due to their larger number of neighbors.

[3]The definition of next-hop atoms can be trivially extended to next-hop routers if more detailed information about ASs is available.

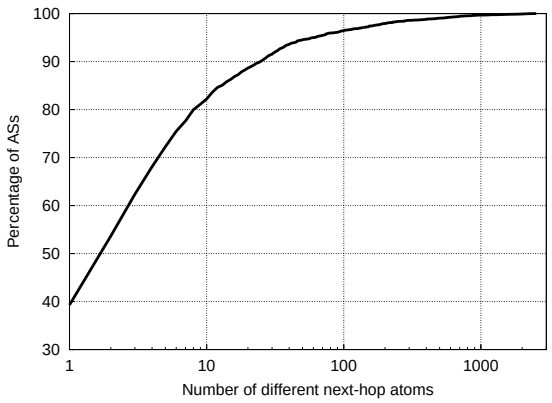

Figure 10: Number of next-hop atoms per AS.

Figure 10 shows the distribution of the number of next-hop atoms per AS, over the 3,535 transit ASs considered in Section 2.1. We observe that about 40% of the 3,535 ASs have a single next-hop atom. Modeling routing policies for those ASs is trivial: they select, for all prefixes, the same set of neighbors. For the remaining 60% of the transit ASs, there can be between a few next-hop atoms up to hundreds. As already mentioned, one expects that the larger the AS, the more diverse its set of path choices, hence the larger its set of next-hop atoms. Figure 11 confirms this belief by giving, for each of the 3,535 transit ASs, the relationship between the number of neighboring ASs and the number of next-hop atoms. A vast majority of the ASs (94%) fall on the $x = y$ line, i.e., have exactly as many next-hop atoms as they have neighbors. Only some highly connected ASs have far more next-hops atoms than neighbors (up to 13 times).

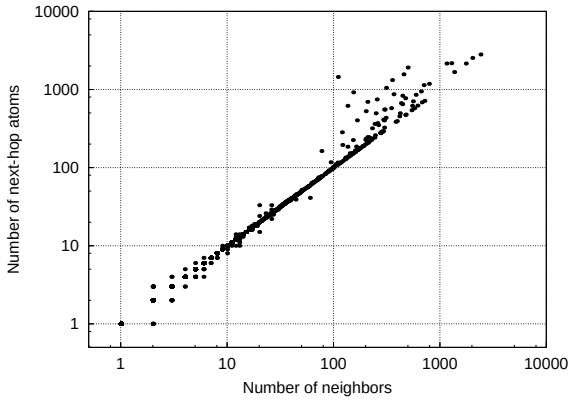

Figure 11: Relationship between number of next-hop atoms and neighbors.

One might conclude from Figure 11 that since the vast majority of ASs have as many next-hop atoms as neighbors, per-neighbor path choices are the rule. This is only true to some extent. Figure 11 does not give any information about how many neighboring ASs any next-hop atom contains. Among all next-hop atoms from our 3,535 transit ASs, more than 75% contain a single neighboring AS (see Figure 12). Only for those next-hop atoms can we configure per-neighbor policies. For the remaining next-hop atoms, preferring a single over all others does not work. In that case, it cannot be only `local-pref` that decides about the choice of the best path, but other rules like MED, IGP cost or other tie-breaking

steps of the BGP decision process. One cannot hope to model such detailed information about path choices by routers, especially by relying only on BGP data from a limited set of vantage points.

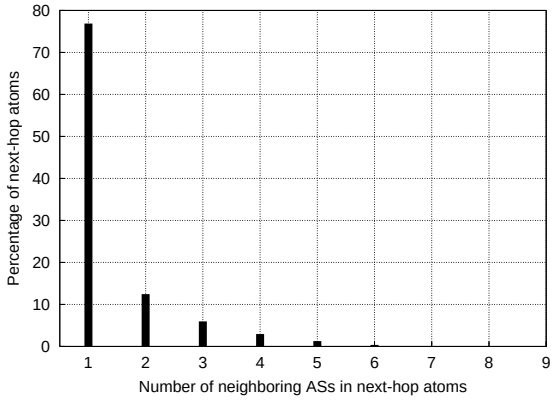

Figure 12: Neighboring ASs in next-hop atoms.

Even though per-neighbor path preferences appear quite common in the Internet, a non-negligible fraction of the path choices are made not by routing policies, but by tie-breaking within the BGP decision process.

To further illustrate the complexity of path choices made by ASs, we study 5 large tier-1 providers in our data. As tier-1 providers have large networks and many neighbors, we would expect them to have complex path choices. Figure 13 provides the number of neighboring ASs in the next-hop atoms of 5 tier-1 providers we selected: UUNET (AS701), AT&T (AS7018), LEVEL3 (AS3356), AOL (AS1668), and OPENTRANSIT (AS5511). We observe huge differences in the fraction of next-hop atoms that are made of a single neighbor (per-neighbor path choices). UUNET has more than 85% of its next-hop atoms consisting of a single neighbor: its path choices are hence very coarse. AOL on the other hand, has less than 5% of its next-hop atoms consisting of a single neighbor. AOL's next-hop atom granularity reflects its business as content provider. AOL is more likely to choose to leverage its path diversity so as to optimize the performance of the paths. OPENTRANSIT is closer to AOL than the other 3 tier-1 providers. UUNET and AT&T have a small fraction of next-hop atoms made of several neighboring ASs. LEVEL3 stands in the middle of those 5 tier-1 providers in the granularity of its path choices.

Modeling how ASs select their path hence depends on the kind of AS being considered. Capturing the full diversity of paths propagated in the Internet, therefore, is not sufficient. We also have to find out what rule of the BGP decision process was used to decide about the path to reach a given prefix. We do not expect this to be an easy task, as it implies inferring very detailed information about AS network engineering.

6. RELATED WORK

Inference of business relationships between ASs [5, 12, 13] has been the most widely studied dimension of routing policies. Routing policies are typically partitioned into a few classes that capture the most common practices in use today [1]. Unfortunately, it is also known that the reality of routing policies [2] and peering relationships is far more complex than those few typical classes [1, 3]. The current approaches for business relationships inference rely on a top-down approach. They first define a set of policies and then try to match those policies with their observations of the system. Yet, policies as used by ISPs have to realize high-level goals [1].

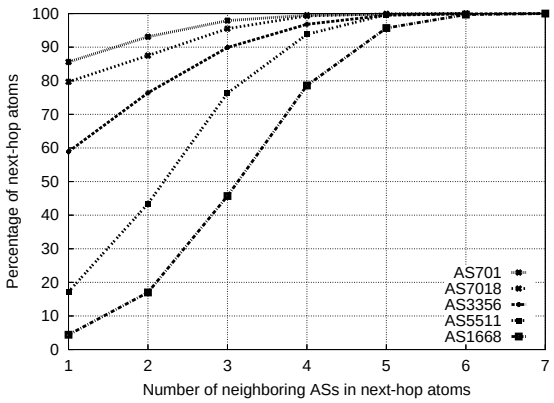

Figure 13: Number of neighboring ASs in next-hop atoms for tier-1 providers.

Assuming any kind of consistency of such policies across ASs is questionable, especially as in practice, policies are often configured on a per-router, per-peering, or per-prefix basis [1]. Observed BGP routes do not have to make those high-level policies visible.

Our work is similar to [7, 14] in allowing the propagation of multiple paths across ASs. The authors in [7] aimed at predicting AS paths between any pair of ASs without direct access to the concerned end-points and relied on a new inference of business relationships, as well as other information to predict the AS paths used between any pair of ASs. [14] showed that to reproduce the diversity of the BGP paths observed from multiple vantage points, it is necessary to allow different routing entities inside each AS to store and propagate the routing diversity known to ASs. Another insight of this paper is that agnosticism about policies in the Internet helps to build a model which is completely consistent with observed BGP data and which has good predictive capabilities. The authors used per-prefix filtering policies to force their model to select the paths observed by BGP.

7. CONCLUSION

In this paper we searched for an appropriate granularity for modeling policies in the Internet. We explored the impact of routing policies on an AS-level model of the Internet. Additionally, we studied how and where to configure policies in this model in such a way that the routes in the model be consistent with paths observed by BGP from multiple vantage points.

By comparing business relationships with per-prefix filters, we investigated the role and limitations of business relationships as a model for policies. We observed there is a large freedom in the location of filters in the model if the goal is to obtain path choices consistent with observed BGP data. We also observed that the popular locations where filtering is necessary in our model correspond to the *valleys* where no path should be propagated according to business relationships inference. This result reinforces the validity of the *valley-free* property used for business relationships inference. However, business relationships do not help to decide which paths among the candidates should be chosen by each AS: after enforcing policies in the model in the form of business relationships, much choice is left as to which route to choose as the best among the candidates. Business relationships do not contain enough information about the path choices made by ASs. To capture the way individual ASs choose their best paths, we introduced a new abstraction: next-hop atoms. Next-hop atoms capture the different sets of neighboring ASs an AS uses for its best routes. We showed that a large fraction of next-hop atoms correspond to per-neighbor

path choices. A non-negligible fraction of path choices however do not correspond to simple per-neighbor preferences, but hot-potato routing and tie-breaking within the BGP decision process, which are very detailed aspects of Internet routing.

The work carried out in this paper provides another step towards a model that may allow prediction of AS paths under "what-if" scenarios. In future work we will validate the policies we derived by testing their predictive capabilities and also by comparing them to actual policies configured by ASs as in [26].

8. ACKNOWLEDGMENTS

We thank Anja Feldmann, our shepherd Dmitri Krioukov and the anonymous reviewers for their comments and suggestions. This work was partially supported by the EU FP6 project DELIS. Bingjie Fu is supported by STW project DTC.6421 and Olaf Maennel by the Australian Research Council (ARC) grant DP0557066. We would also like to thank Belinda Chiera for improvements in the writing and TU München for providing computational infrastructure for our simulations.

9. REFERENCES

[1] M. Caesar and J. Rexford, "BGP Routing Policies in ISP Networks," *IEEE Network Magazine*, 2005.

[2] F. Wang and L. Gao, "Inferring and Characterizing Internet Routing Policies," in *Proc. ACM IMC*, 2003.

[3] H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Towards Capturing Representative AS-Level Internet Topologies," *Computer Networks*, vol. 44, no. 6, April 2004.

[4] T. Griffin, F. Bruce Shepherd, and G. Wilfong, "The Stable Paths Problem and Interdomain Routing," *IEEE/ACM Trans. Networking*, 2002.

[5] L. Gao, "On Inferring Autonomous System Relationships in the Internet," *Proc. IEEE Global Internet*, 2000.

[6] Z.M. Mao, J. Rexford, J. Wang, and R.H. Katz, "Towards an Accurate AS-level Traceroute Tool," in *Proc. ACM SIGCOMM*, 2003.

[7] Z.M. Mao, L. Qiu, J. Wang, and Y. Zhang, "On AS-level Path Inference," in *Proc. ACM SIGMETRICS*, 2005.

[8] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of Hot-Potato Routing in IP Networks," in *Proc. ACM SIGMETRICS*, 2004.

[9] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan, "Traffic Matrix Reloaded: Impact of Routing Changes," in *Proc. PAM*, 2005.

[10] R. Teixeira, K. Marzullo, S. Savage, and G. Voelker, "In Search of Path Diversity in ISP Networks," in *Proc. ACM IMC*, 2003.

[11] N. Feamster, Z. Mao, and J. Rexford, "BorderGuard: Detecting Cold Potatoes from Peers," in *Proc. ACM IMC*, 2004.

[12] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the Internet Hierarchy from Multiple Vantage Points," in *Proc. IEEE INFOCOM*, 2002.

[13] G. Battista, M. Patrignani, and M. Pizzonia, "Computing the Types of the Relationships between Autonomous Systems," in *Proc. IEEE INFOCOM*, 2003.

[14] W. Mühlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig, "Building an AS-Topology Model that Captures Route Diversity," in *ACM SIGCOMM*, 2006.

[15] "RIPE's Routing Information Service," `http://www.ripe.net/ris/`.

[16] "University of Oregon RouteViews Project," http://www.routeviews.org/.

[17] Intel-DANTE, "Intel-DANTE Monitoring Project," http://www.cambridge.intel-research.net/monitoring/dante/.

[18] Abilene, "The Abilene Observatory: Abilene routing data," http://abilene.internet2.edu/observatory/.

[19] S. Uhlig and S. Tandel, "Quantifying the Impact of Route-Reflection on BGP Routes Diversity inside a Tier-1 Network," in *Proc. of IFIP Networking*, Coimbra, Portugal, May 2006.

[20] Y. Afek, O. Ben-Shalom, and A. Bremler-Barr, "On the Structure and Application of BGP Policy Atoms," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 2002.

[21] A. Broido and KC. Claffy, "Analysis of RouteViews BGP Data: Policy Atoms," in *Proceedings of the Network-Related Data Management workshop, Santa Barbara*, May 2001.

[22] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. Felix Wu, and L. Zhang, "An Analysis of BGP Multiple Origin AS (MOAS) Conflicts," in *Proc. ACM IMW*, 2001.

[23] J. Hao M. Meulle, Q. Nguyen, "Formulation CSP et Approches Heuristiques pour l'Inférence des Accords d'Interconnexion dans l'Internet," in *ROADEF'06*, Lille, France, 2006.

[24] B. Quoitin and S. Uhlig, "Modeling the Routing of an Autonomous System with C-BGP," *IEEE Network Magazine*, 2005.

[25] B. Quoitin, "C-BGP, an Efficient BGP Simulator," http://cbgp.info.ucl.ac.be/, 2003.

[26] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, kc claffy, and G. Riley, "AS Relationships: Inference and Validation," *ACM Comput. Commun. Rev.*, vol. 37, no. 1, 2007.

[27] G. Di Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and T. Schank, "Computing the Types of the Relationships between Autonomous Systems," *To appear in IEEE/ACM Transactions on Networking*, 2007.

APPENDIX

A. RELATIONSHIP INFERENCE

Previous work on inference of AS business relationships (e.g., [5]) relies on two main assumptions. First, there is a unique business contract negotiated between any two ASs. The relationship associated with any directed link of the AS topology is one of the following: peer-peer, customer-to-provider (C2P) or provider-to-customer (P2C)[4]. Second, routes of an AS received from any of its provider or peer-peer neighbor cannot be propagated to any other provider or peer-peer neighbor. In the literature, this is called the "valley-free" property.

Inferring business relationships can be formulated as the MaxTOR problem [12]. Given a set of AS paths, assign a unique label to each link in the AS topology such that the number of valley-free paths is maximum. MaxTOR is a NP-complete problem that was tackled in the past by many heuristics because of its size. Moreover, as reported in [27], if a solution to MaxTOR with n business relationships exists (optimal or not), then there are at least 3^n dif-

[4]two ASs may also establish a mutual-transit relationship (sibling or *SIB*)

ferent solutions leading to the same number of valley-free paths. This means that any given solution has a very small probability of being realistic, even if it produces the exact maximum number of valley-free paths.

As inferred relationships can produce up to 99% of observed valley-free paths, the propagation of observed paths remains possible in an AS topology model when relationship filters are incorporated. However, as there is an exponential number of solutions with the same number of valley-free paths, a large freedom remains in the choice of relationship filters. Many approaches tackled the MaxTOR problem by splitting each path into AS triples. From the several approaches, we selected four:

- **gao [5]:** The greedy approach recognizes valley-free AS paths by using sequences of AS degrees in paths. When valleys occur in AS paths, some conflicting relationships are supposed to support a mutual-transit relationship (label "SIB").
- **sark [12]:** This algorithm uses topology leaf-pruning as seen from each observation point to infer per-vantage-point AS rankings. Then a relationship for each link is inferred. When ranking of ASs is not decisive enough, some links are labeled with the "Unknown" relation.
- **csp [23]:** This approach takes advantage of the Constraint Satisfaction framework. A Max2CSP problem is derived from MaxTOR where each relation is a variable and each sub-path of length 2 (AS triples) introduces a constraint between two relations. A tabu-search algorithm runs on a restricted space of feasible solutions (unlikely relations and customer cycles are forbidden).
- **caida [26]:** Another recent algorithm claims to find more realistic solutions with a partial validation of the results. The objective function to maximize in the MaxTOR problem is modified to incorporate information on degree of ASs. This mathematical program is solved by using Semi-Definite Programming and then uses a post-processing heuristic that tries to maximize the number of peering links.

To compare the four algorithms *gao*, *sark*, *csp* and *caida*, we use as input our full dataset (labeled *(1)*) and subsets of AS paths gathered at Route-views and RIPE (labeled *(2)* and *(3)*). For each dataset, we report in Table 6 the size of the AS topology, the number of distinct AS paths of length 2 (triples) and the size of the MAX2CSP models solved by the *csp* algorithm. Note that AS paths of length 1 (2 AS hops) are always valley-free. In particular, relationships supported by links observed only in paths of length 1 can be treated separately and removed from the input data (indicated as *no-cons* in the table).

We evaluate the number of valley-free paths and the number of valley-free AS triples for each of the 4 inference algorithms and for the 3 input datasets in Table 7. Note that we were not able to run the *caida* algorithm on our data and therefore downloaded an existing solution from CAIDA for November 7th, 2005 (solution only based on traditional RIPE and Route-Views data). For each solution, we report the number of peer-peer links and customer-provider links inferred, as well as the number of *Unknown* or *mutual-transit* links. Also, to obtain an understanding of the correctness of the solutions, we validate the inference with two indicators: *Caida-match* and *Tier-1-match*. *Caida-match* is the percentage of relationships that are inferred as being of the same type by both the considered heuristic on our data and the downloaded *caida* solution. *Tier-1-match* is the percentage of relationships that are of the same type by both the considered heuristic on our data and those we know to be the real relationships for a tier-1 in November 2005.

Our full dataset has twice as many unique AS paths as the RIPE or Route-views subsets, but approximately the same number of ASs and only 20% more AS relationships. Altogether, it contains roughly twice as many sub-paths of length 2 (AS triples) as the two subsets. Additional input paths increase the difficulty of MaxTOR and make it more restrictive on potential solutions. Still, many solutions maximizing the same number of valley-free paths exist. We now evaluate the solutions produced by the different algorithms.

Paths	AS graph		CSP models		
all	vertices	links	triples	variables	no-cons
Full dataset (1)					
4 681 770	21 169	58 911	965 859	54 193	4 718
RIPE subset (2)					
1 972 727	21 016	48 162	415 523	46 489	1 673
Route-views subset (3)					
1 682 568	21 060	47 170	476 668	45 197	1 973

Table 6: Datasets used to infer business relationships.

	relations			**valley-free**		**match**	
	PEER	*C2P*	*SIB/ UNK*	triples	paths	*CAIDA*	*Tier-1 match*
sark heuristic							
1	25688	32703	520	81.5	27.3	54.8	84.2
2	13786	34006	370	84.6	29.9	66.3	82.7
3	15630	31188	352	85.5	32.8	61.3	85.7
gao heuristic							
1	12971	44252	1688	88.3	100.0	92.6	65.4
2	5200	41453	1509	90.6	100.0	93.5	66.9
3	5361	40333	1476	90.5	100.0	94.2	69.2
caida heuristic							
1	3367	38128	229	70.5	96.1	100.0	80.0
2	3367	38128	229	73.0	96.3	100.0	80.0
3	3367	38128	229	85.4	97.4	100.0	80.0
csp heuristic							
1	18326	40585	0	99.9	99.3	95.0	94.7
2	9219	38943	0	99.9	99.3	95.5	94.0
3	8050	39120	0	99.9	99.3	94.5	94.7

Table 7: Evaluation of solutions provided by algorithms.

The *sark* algorithm produces a small number of valley-free paths (see Table 7). However, the solutions match well the relationships from the tier-1 (about 80%). The *gao* heuristic has 100% of valley-free paths. Some relationships are set to *mutual-transit*, having as effect to cancel valleys next to this kind of links. This algorithm determines an unrealistically large number of *mutual-transit* relationships. Note that solutions match well the relationships of the tier-1 (about 65%). The *caida* solution was not produced using our datasets. Its evaluation on each dataset produces a large number of valley-free paths (more than 96% of our paths are valley-free) and a good match with the relationships of the tier-1 (80%). Finally, the *csp* heuristic produces solutions with the largest number of valley-free paths (up to 99%) and the best match with the relationships of our tier-1 (about 94%). Even if solutions produced by any algorithm have a large number of valley-free AS triples, the remaining freedom in the choice of relationships does not favor all algorithms. Only few paths are valley-free or the values of our *match* indicators (of realism) are not high enough. Indeed, for the three datasets, only solutions provided by the *csp* and *gao* algorithms are close to the *caida* one. Since some solutions of the *caida* algorithm have been validated using information about business relationships of many ASs, we consider those produced by *csp* as realistic enough for our goal. We thus rely on the *csp* algorithm to run our model of path propagation, because of its potential accuracy and its large number of valley-free paths.