Towards a Semantic Framework for Policy Exchange in the Internet

Anduo Wang
Temple University, Philadelphia, USA
adw@temple.edu

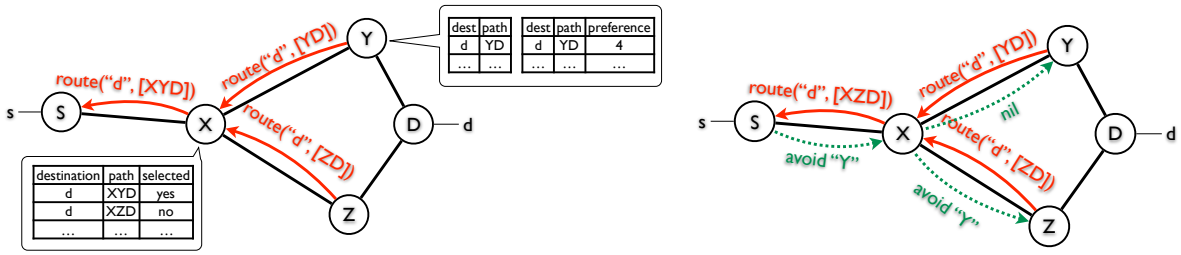
The major issue addressed in *autonomous* data source management, notably within the framework of data exchange, is the variety of heterogeneous data schemas — the *forms* the data must take. But semantics that give meaning to the factual data can be autonomous as well. In certain context such as Internet routing that lacks a central authority, both the data item — distributed routing tables that collectively constitute a global path, and the semantic data — polices that determine the selection of routing path, are independently managed by networks (autonomous systems (ASes)) participating in the Internet. A long standing networking problem is inflexible policy routing due to the lack of policy sharing mechanisms. This paper seeks to extend the classic data exchange framework to semantic data as a means to manage distributed, autonomous routing polices in the Internet. Specifically, we make the case of knowledge exchange, and use policy exchange as a driving problem — a special case — to understand the semantic framework of knowledge exchange: We identify several unique challenges, and illustrate how answer set programming — originally developed for knowledge representation with negation and non-monotonic reasoning, and residue method based transformation — originally developed for semantic query optimization, may serve as a point solution to the specific policy exchange problem.

1 A long standing networking problem: policy routing

In the absence of a central authority, the Internet is a loose confederation of networks, also called domains or autonomous systems (ASes) [13]. Each AS manages a subset of Internet resources — routers, transmission links, and hosts etc. To allow communication between any two hosts, potentially located at two distant ASes, the routing function is employed to find a path, a sequence of ASes from the source to the destination. During distributed route computation, as shown in Figure 1 (a), each AS independently performs the following: Each AS receives one or more routes from its direct neighbors. A route is a record that contains a destination address and several attributes (e.g., AS path), it is the sender's offer to carry traffic to the destination along the AS path in the route. For all the received routes (to a certain destination), a single best route is selected and announced to other neighbors.

A unique requirement of routing in the Internet is the need to support AS policy, enabling individual ASes to influence global path selection with their own interests [9, 14]. Consider economic policies based on business relations for example: Smaller ASes (customer) often subscribe to larger ASes (provider) to obtain Internet access. A provider AS may wish to only allow certain traffic to traverse its facilities; Similarly, a customer AS may prefer to route traffic through a designated provider. To realize such policies, the Internet routing architecture employs selective route distribution and independent route selection: A provider AS can propagate routing data to a customer only if it is willing to receive traffic from that customer; A customer AS can steer traffic towards a particular provider by independent route selection, assigning highest preference to routes from the preferred provider. These policy mechanisms,

Submitted to:
LFMTP 2021



(a) distributed route computation (b) logic sentence propagation

Figure 1: Routing data exchange as distributed route computation: X,Y,Z,S,D are ASes; s and d are the source and destination hosts.

implemented by today's one single de-facto inter-domain routing protocol (border gateway protocol, or BGP [13]), are widely used in commercial Internet for more than two decades.

Yet, the BGP policy mechanisms are known to be extremely inflexible. Consider again AS S as shown in Figure 1 (b), suppose S subscribes to AS X to reach the rest of the Internet, and considers some downstream (closer to destination) AS Y suspicious. What AS X demands is a path avoiding Y to d. Yet, the failure to disclose this constraint to X can has unfortunate consequence: not knowing S's constraint, X might propagate the "invalid" path XYD to S, even though an acceptable path does exist. The root cause here is that opaque policy can severely restrict an AS's visibility into the routing space, inherently excluding otherwise realizable policies. On the other hand, a customer does have incentive to disclose its local constraints as a form of explicit routing service request, as shown in Figure 1 (b). After all, an AS cannot expect certain properties of the routes it can obtain from a neighbor if it does not announce what property it desires in the first place.

Problem: Current routing architecture does not provide any support for policy exchange. While home grown mechanisms driven by specific problems were proposed to improve policy visibility, a general solution is still missing. Can we borrow from decades of data exchange and knowledge representation research to derive a principled solution?

2 New opportunities and challenges: policy exchange

Data exchange [4, 11, 3] offers well tested theory and tool for shipping information from one source to another, but it does not fit the policy exchange problem out of the box. To see why, we first review the data exchange formulation. Essentially, data exchange asks, as shown in Figure 2 (left), given a source schema X, a database instance I over X, target schema Y, constraint Σ_{XY} that defines the relations between X and Y, and (optionally) additional constraint Σ_Y imposed at Y, what is the instance J over Y that can simultaneously satisfy the constraints Σ_{XY} and Σ_Y. The focus is factual data transformation and query answering over the transformed facts. But routing policies are semantic data — the "avoid AS Y" is an integrity constraint (IC) that must be satisfied for the routing table.

To extend data management support to semantic data like policies, we propose an extension to data exchange, which we term as knowledge exchange. As shown in Figure 2 (right), given two data instances I, J over schema X, Y, respectively, the relationships between X, Y as specified by Σ_{XY}, and ICs Σ_X over X, find Σ_Y over Y that, to ensure consistency of I — compliance with Σ_X, it suffices to ensure consistency of J with respect to Σ_Y.

The policy exchange problem in Figure 1 (b) can be viewed as a special case of knowledge exchange:

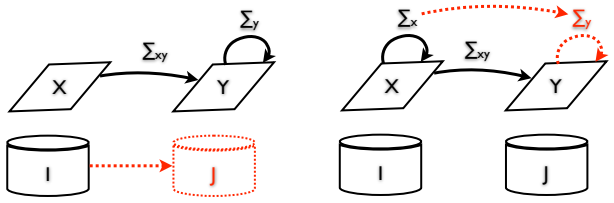


Figure 2: (left) routing data exchange as classic data exchange; (right) routing policy exchange calls for new technique — knowledge exchange.

the data instances are the routing data independently managed by each AS, potentially with different schemas as shown in Figure 1 (a). Relationships between these routing schemas are defined by the distributed route computation — in fact, current routing architecture serves as a data exchange facility for propagating routing data from ASes closer to the destination to those towards the source. Policy exchange asks, on the other hand, given a policy IC defined over schema at S, what is the right policy IC to be imposed at its neighbor X such that given the routing data dependency between S and X — as routes at S are learned from X — the transformed policy at X will guarantee consistency at S.

This new knowledge exchange setup creates new data challenges. As a first step to understand this problem, in the following, we use policy routing in the Internet as a concrete special case to discuss several unique challenges:

- What is the good representation for policy ICs (Σ_X, Σ_Y) and the schema dependencies (Σ_{XY})? In practical database systems, ICs are often kept simple for performance concerns. Even when considering ICs in classic logic, they are monotonic by nature — once a fact is derived, it cannot be removed as new data items arrive. But routing policies are non-monotonic by nature: a new path can take over a path previously preferred. Besides, routing policies, unlike pure logic statements, are not "equal", a more important policy can override a less important one. Can we find a representation language that is both efficient and expressive? In § 3.1, we illustrate the use of answer set programming (ASP) as a high-performance knowledge representation for non-monotonic policies. We also introduce new concept and method for prioritizing policies.
- Can we develop efficient procedure that can take the knowledge exchange problem $(X, Y, \Sigma_{xy}, \Sigma\Sigma_X)$ as input, and compute the desirable Σ_Y? What is the formal semantics of the target policy? Does a solution — target policy that satisfy the problem — always exist? As a first step to answering these questions, we develop customized transformation procedure for the special policy exchange problem. In § 3.2 we identify two scenarios of the problem, and, by view unfolding and borrowing from semantic query optimization, we develop IC transformation procedures for both scenarios.

3 Preliminary Design

This section sketches a preliminary design of policy exchange.

3.1 Policy Representation

We adopt answer set programming (ASP) [1, 5] as the knowledge representation for routing polices. ASP statements resemble a datalog rule, with the exception that it provides native support for negation with stable model semantics [6, 7]. The native support for negation makes ASP particularly convenient for non-monotonic routing policies.

In an ASP rule like the following:

```
a :- b1,...bm, not c1,.., not cn.
```

Each predicate `a, bi` is either an atomic formula or the negation of the atomic formula. The rule itself establishes (or derives) that `a` (also called the *head* of a rule) is true if all the non-negated predicates `bi` is presented but the negated predicates `cj` are not (these predicates on the right hand side of `:-` are also called the *body*).

Basic representation: polices as ASP rules

Consider the "avoid AS" policy in § 1 as an example. To encode this policy, we introduce two predicates `r` and `p` for routing data: `r(p,d)` states that route `p` is permitted to reach destination prefix `d`; `b(p,d)` says that `p` is selected as the best route to get to `d`; and `p` is a tuple that contains the list of ASes along the path, and `d` is a destination prefix. For example `r((120),'1.2.3.4')` and `b((20),'1.2.3.4')` represent a candidate path via ASes `(120)` to prefix `'1.2.3.4'` and a selected best path via `(20)`, respectively. Based on these predicates, we have the following denial rule `ps`:

```
% security constraint
ps :- b(p,d), waypoint(p,'x').
```

As a second example, consider the well known Gao-Rexford policy that prefers a customer route over a provider route. We draw on the insight that a path `p1` is more preferred over a path `p2` if `p2` can be promoted to be the best path only when `p1` is not presented:

```
% p1 is preferred to p2 if p1 is customer route and p2 is a provider route
pg1 b(p1,d) :- r(p1,d), customer(p1).
pg2 b(p2,d) :- r(p2,d), not r(p1,d), customer(p1), provider(p2).
```

Policies with priorities

With the standard notion of ASP rules — classic logic statements in general [8, 12], a subtle but important limitation is that, all the policies are considered "equally important". In the presence of multiple policies, if they cannot be simultaneously satisfied on an AS's current routing data, it has either accept the result of no path to a destination, or ignore some policies. In networking, however, a more sensible approach is to explicitly support priorities, allowing policies that can override one another, where the impact of a less important policy may be restricted by a more important one.

For example, consider an AS with two policies: an economic policy that prevents the AS from picking routes from a provider if a cheaper customer route is available, and a security policy that attempts to avoid a certain suspicious AS X. Also suppose that the AS has a provider (cheap) path that waypoints X (insecure) and a customer path (expensive) that avoids X (secure). A path that is jointly cheap and safe does not exist. Rather than concluding that no usable path is available, with security as a higher priority, a more sensible alternative is to give the security policy higher priority. That is, the economic policy will take effect only among multiple paths that are safe. In this example, a routing table with the expensive but secure path — provider path avoiding X — will be considered consistent.

But how can we express this notion of priority within standard logic? To illustrate the connection to standard logic, consider a second example: Suppose a domain A has three candidate paths $\{AXD, ABXD, ABCD\}$ to a destination D, and manages two policies: a high priority security policy — *ps* as discussed in § 3.1, and a low priority policy — a new shortest path policy (*psp*) that eliminates longer paths:

```
% shortest path constraint: a best path cannot be longer than any existing
```

```
   paths, has a lower priority
psp (ps<psp) :- b(p1,d), r(p1,d), r(p2,d), length(p1) > length(p2).
```

With a lower priority, *psp* will not eliminate *ABCD* even though it is longer than *AXD*, because *AXD* does not satisfy the higher ranked *ps* constraint. In other words, a path can be eliminated by *psp* if it jointly satisfies (1) the *psp* violation condition; and (2) the *ps* requirement. This intuition can be captured by the "normal" denial constraint psp' in the following:

```
% shortest path constraint restricted to path compliant with the security
   constraint.
psp' :- b(p1,d), r(p1,d), r(p2,d), length(p1) > length(p2), not
   waypoint(p1,'X'), not waypoint(p2,'X').
```

Here, priority is captured by the transformation from *psp* to psp'.

Normalizing priority with semantic transformation

To automate the normalization process, we develop an automated transformation process as follows. For every two denials p_i, p_j with a priority assignment $p_i < p_j$, to remove the priority assignment, we only need to semantically constrain p_j by p_i; thereby producing the semantically constrained version of p_j, written as p'_j, while keeping p_i as it is. To this end, we leverage the residue method developed for semantic query optimization [2, 10, 8].

Key to the original residue method is the notion of residue, a fragment of an integrity constraint that anticipates its impact on a relation. Residue is a syntactic fragment that can be obtained by the standard subsumption algorithm — in the case of partial subsumption of a relation by the integrity constraint, residue is the fragment of the integrity constraint that remains at the bottom of the refutation tree. For example, the residue of an integrity constraint (`:- b(p,d), length(p) < 3.`) over $b(p,d)$ is `:- length(p) < 3.`, meaning that the presence of `b(p,d)` implies `not (length(p)<3)` (i.e.`length(p)>= 3`).

The residue `not (length(p)<3)` of (`:- b(p,d), length(p) < 3.`), when attached to $b(p,d)$, allows us to incorporate the integrity constraint, and to transform a query involving $b(p,d)$ into an equivalent form: Intuitively, any query with b must satisfy the condition stated by the residue to succeed. For example, `?:- b(p,'1.2.3.4'), length(p) = 2` — a query asking for a 2-hop path to `'1.2.3.4'`, is semantically transformed to `?:- b(p,'1.2.3.4'), length(p) = 2, length(p) >= 3.`. The transformed query contains a logical contradiction, thus can be easily determined without actually processing the query. In general, the benefit of the residue method is that, explicitly associating the integrity constraint with a relation reduces the search space of integrity constraints that can apply to a given query.

Similarly, we apply residue annotation to integrity constraints themselves, transforming the annotated integrity constraint into a constrained form. For example, the residue of *ps* over $r(p1,d)$ and $r(p2,d)$ is `not waypoint(p1,'X')` and
`not waypoint(p2,'X')`, respectively. Attaching both to *psp* gives the semantically constrained psp', which is equivalent to *psp* that yields to *ps* — i.e. *psp* with the priority assignment $ps < psp$.

3.2 Policy Exchange

Specifying policy exchange

First, we capture the relationships between routing schemas at different ASes by the AS topology. The intuition is simple: a routing data item is always an "path extension" of data item at some neighboring AS closer to the destination of the path. In the special case of policy exchange, the schema mapping

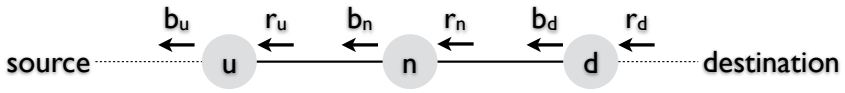


Figure 3: Schema mapping as topology constrains: a permitted route at an AS always correspond to the (selected) best route at some neighboring node.

Σ_{XY} in Figure 2 become tuple generating dependency based on topology, which can be conveniently encoded in ASP.

Consider the AS topology in Figure 3 as an example: without loss of generality, we also fix the source and destination and omit destination d in the ASP coding: as routes to d (originated as `rd`) are propagated from the destination towards the upstream (`ru`, closer to the source), the AS policy of each node is used to select the best routes (`bu,bn,bd`).

```
% link constraints
lu ru(p,d) :- bn(p',d), p = u o p'.
ld rn(p,d) :- bd(p',d), p = n o p'.
```

Here, `o` is the concatenation operator that extends a path with a new node. Rule `lu` specifies the link connecting node `n` to its upstream neighbor `d` that a (selected) route announced on that link will result in a permitted path for that neighbor. Similarly, rule `ld` is the constraint of the upstream link that ensures any permitted path of node `n` must be learned from some downstream neighbor.

In addition, a selected route must correspond to one of the permitted routes. In other words, the occurrence of a selected path `b(p,d)` in the absence of `r(p,d)` should signal an error. We write such a constraint as a headless ASP rule of the form `:- b1,...bm, not c1,.., not cn.` (`bi` and `cj` are predicates), which conveys the intuition that satisfying `B` results in a contradiction.

```
% node constraint
n :- bn(p,d), not rn(p,d).
```

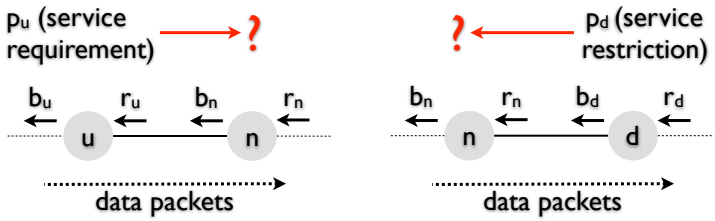


Figure 4: (left) propagate service requirement to the downstream neighbors; (right) propagate service restriction to the upstream neighbors.

Solving policy exchange

To derive a solution to the policy exchange problem — target policy ICs Σ_Y at Y (Figure 2), we develop a semantic transformation process — a combination of view unfolding and the residue method. In policy routing, as shown in Figure 4, we only need to consider two specific scenarios: (1) For policies at a upstream AS representing route requirement that cannot be enforced locally, compute the corresponding polices that must be satisfied at the downstream too meet the upstream's original request; (2) For polices

at a downstream AS encoding a non-trivial service restriction, compute what restriction must be enforced at an upstream neighbor to honor the original upstream restriction.

Due to limited space, in this short paper, we only present the transformation process for (1) using an abstract policy (service requirement) *pu* of a upstream domain *u* in Figure 4 (left):

```
pu :- bu(p,d), cu(p).
```

`pu` specifies what must be excluded from best routes, with `cu` capturing what is undesirable. The security constraint `ps` in § 3.1 is an instance of this policy. Using this abstract policy as an example, we illustrate how to project it into a policy local to `n`, denoted by `Pn`. We note that `Pu` is a constraint over `bu` whereas `Pn` must be a constraint over `bn`. Based on this observation, our strategy is to make the transformation in two steps: first, leverage the knowledge in the node constraint of `u` to translate constraints over `bu` onto that over `ru`; next, use the edge constraint (for the link between `u` and `n`) to map constraints over `ru` to `bn`.

To apply the node constraint `u :- bu(p), not rn(p).`, we leverage the residue method, a knowledge base technique that infers the impact of a constraint with the theorem proving technique called partial subsumption. The idea is that there exists a subsumption algorithm that computes a fragment of the constraint, called residue, that anticipates its impact. Here, the residue of `u` with respect to `Pu` is `:- not {ru(p)}`, or, equivalently, `{ru(p)}`. Intuitively, it means every occurrence of `ru(p)` should be accompanied by the `ru(p)`. Thus, we transformed `pu` to `t1`:

```
t1 :- bu(p), cu(p), ru(p)
```

Assume that any permitted route (`ru(p)`) may be selected as the best route — we call this the "import axiom", i.e. the policy system itself does not preclude any local policy, we can safely drop `bu(u)` from the above constraint and derive `t2`:

```
t2 :- cu(p), ru(p).
```

Next, we apply the edge constraint `lu` (§ 3.1) to map `t2` over onto `bn` by constraint composition: the effect of `t2` and `lu` (`ru(p,d) :- bn(p',d), p = u o p'.`) is equivalent to `pn`:

```
% abstract requirement exposed to n
pn :- cu(p), bn(p',d), p = u o p'.
% concrete requirements exposed to n after simplification
pn1 :- waypoint(p','x'), bn(p',d).
pn1 :- length(p')>2, bn(p',d).
```

`pn` relates `cu` to the best path of `n` (`bn`), and thus characterizes the policy information that should be exposed to `n`. We can also instantiate this abstract requirement: if `cu(p) = waypoint(p,'x')`, then we can derive `pn1` by simplifying `waypoint(p,'x')/\p = u o p'.` to `waypoint(p','x').` As a second concrete example, if
`cu(p) = length(p)>3` meaning that node `u` requires a path of less than three hops to reach the destination, it will map into a constraint that requires that downstream `n` to find a path with at most 2 hops.

4 Conclusion

We call for advancement in logic reasoning and semantic framework community to take an active role in managing policies for the Internet. Specifically, by going beyond the traditional treatment of routing

policies as properties of the routing data, we lift policies to be first-order data that can be explicitly exchanged, and propose a novel extension to traditional data exchange framework to handle semantic data — network policies. We hope that, with some luck, the proposed knowledge-driven technique enablers may infect the design and migrate into the fabric of the future network system.

References

[1] BREWKA, G., EITER, T., AND TRUSZCZYNSKI, M. Answer set programming at a glance. *Commun. ACM 54*, 12 (Dec. 2011), 92–103.

[2] CHAKRAVARTHY, U. S., GRANT, J., AND MINKER, J. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst. 15*, 2 (June 1990), 162–207.

[3] FAGIN, R., FAGIN, R., FAGIN, R., KOLAITIS, P. G., AND POPA, L. Data exchange: Getting to the core. *ACM Trans. Database Syst. 30*, 1 (Mar. 2005), 174–210.

[4] FAGIN, R., KOLAITIS, P. G., MILLER, R. J., AND POPA, L. Data exchange: Semantics and query answering. *Theor. Comput. Sci. 336*, 1 (May 2005), 89–124.

[5] GEBSER, M., KAUFMANN, B., KAMINSKI, R., OSTROWSKI, M., SCHAUB, T., AND SCHNEIDER, M. Potassco: The potsdam answer set solving collection. *Ai Communications 24*, 2 (2011), 107–124.

[6] GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. MIT Press, pp. 1070–1080.

[7] GELFOND, M., AND LIFSCHITZ, V. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9* (1991), 365–385.

[8] GODFREY, P., GRANT, J., GRYZ, J., AND MINKER, J. Integrity constraints: Semantics and applications. In *Logics for Databases and Information Systems (the book grow out of the Dagstuhl Seminar 9529: Role of Logics in Information Systems, 1995)* (1998), pp. 265–306.

[9] GRIFFIN, T. G., JAGGARD, A. D., AND RAMACHANDRAN, V. Design principles of policy languages for path vector protocols. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2003), SIGCOMM '03, ACM, pp. 61–72.

[10] KING, J. J. *Query Optimization by Semantic Reasoning*. PhD thesis, Stanford, CA, USA, 1981. AAI8124097.

[11] KOLAITIS, P. G. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2005), PODS '05, ACM, pp. 61–75.

[12] REITER, R. On integrity constraints. In *Proceedings of the 2Nd Conference on Theoretical Aspects of Reasoning About Knowledge* (San Francisco, CA, USA, 1988), TARK '88, Morgan Kaufmann Publishers Inc., pp. 97–111.

[13] REKHTER., Y., LI., T., AND HARES., S. A Border Gateway Protocol 4 (BGP-4).

[14] WANG, A., CHEN, Z., YANG, T., AND YU, M. Enabling policy innovation in interdomain routing: A software-defined approach. In *SOSR* (2019).