Modeling Complexity of Enterprise Routing Design

Xin Sun
School of Computing and
Information Sciences
Florida International University
xinsun@cs.fiu.edu

Sanjay G. Rao
School of Electrical and
Computer Engineering
Purdue University
sanjay@purdue.edu

Geoffrey G. Xie
Department of Computer
Science
Naval Postgraduate School
xie@nps.edu

ABSTRACT

Enterprise networks often have complex routing designs given the need to meet a wide set of resiliency, security and routing policies. In this paper, we take the position that minimizing design complexity must be an explicit objective of routing design. We take a first step to this end by presenting a systematic approach for modeling and reasoning about complexity in enterprise routing design. We make three contributions. First, we present a framework for precisely defining objectives of routing design, and for reasoning about how a combination of routing design primitives (e.g. routing instances, static routes, and route filters etc.) will meet the objectives. Second, we show that it is feasible to quantitatively measure the complexity of a routing design by modeling individual routing design primitives, and leveraging configuration complexity metrics [5]. Our approach helps understand how individual design choices made by operators impact configuration complexity, and can enable quantifying design complexity in the absence of configuration files. Third, we validate our model and demonstrate its utility through a longitudinal analysis of the evolution of the routing design of a large campus network over the last three years. We show how our models can enable comparison of the complexity of multiple routing designs that meet the same objective, guide operators in making design choices that can lower complexity, and enable what-if analysis to assess the potential impact of a configuration change on routing design complexity.

1 Introduction

Recent studies [16, 20] show that routing designs of many enterprise networks are much more complicated than the simple models presented in text books and router vendor documents. Part of the complexity is inherent, given the wide range of operational objectives that these networks must support, to include security (e.g., implementing a subnet level reachability matrix), resiliency (e.g., tolerating up to two component failures), safety (e.g., free of forwarding loops), performance, and manageability. There is also evidence, however, to suggest that some of the network design complexity may have resulted from a semantic gap between the high level design objectives and the diverse set of routing protocols and low level router primitives for the operators to choose from [23]. Often, multiple designs exist to meet the same operational objectives, and some are significantly easier to implement and manage than others for a target network. For example in some cases, route redistribution may be a simpler alternative to BGP for connecting multiple routing domains [16]. Lacking an analytical model to guide the operators, the current routing design process is mostly ad hoc, prone to creating designs more complex than necessary.

In this paper, we seek to quantitatively model the complexity associated with a routing design, with a view to developing alternate routing designs that are less complex but meet the same set of operational objectives. Quantitative complexity models could enable systematic abstraction-driven top-down design approaches [23], and inform the development of clean slate network architectures [9, 13], which seeks to simplify the current IP network control and management planes.

The earliest and most notable work on quantifying complexity of network management was presented by Benson et al. [5]. This work introduced a family of complexity metrics that could be derived from router configuration files such as dependencies in the defintion of routing configuration components. The work also showed that networks with higher scores on these metrics are harder for operators to manage, change or reason correctly about.

While [5] is an important first step, it takes a bottom-up approach in that it derives complexity metrics from router configuration files. This approach does not shed direct light on the intricate top-down choices faced by the operators while designing a network. Conceivably, an operator could enumerate all possible designs, translate each into configurations, and finally quantify the design complexity from the configurations. However, such a brute-force approach may only work for small networks where the design space is relatively small. Additionally, this approach still requires a model to determine which designs actually are correct, i.e., meeting the design objectives.

In this paper, we present a top-down approach to characterizing the complexity of enterprise routing design given only key high-level design parameters, and in the absence of actual configuration files. Our model takes as input abstractions of high-level design objectives such as network topology, reachability matrix (which pairs of subnets can communicate), and design parameters such as the routing instances [20] (see Section 2 for formal definition), and choice of connection primitive (e.g., static routes, redistribution etc). Our overall modeling approach is to (i) formally abstract the operational objectives related to the routing design which can help reason about whether and how a combination of design primitives will meet the objectives; and (ii) decompose routing design into its constituent primitives, and quantify the configuration complexity of individual design primitives using the existing bottom-up complexity metrics [5].

A top-down approach such as ours has several advantages. By working with design primitives directly (independent of router configuration files), the model is useful not only for analyzing an existing network, but also for "what if" analysis capable of optimizing the design of a new network and similarly, a network migration [24], or evaluating the potential impact of a change to network design. Further, our models help provide a conceptual framework to understand the underlying factors that contribute to configuration complexity. For example, reachability restrictions between subnet pairs may require route filters or static routes, which in turn manifest as dependencies in network configuration files.

We demonstrate the feasibility and utility of our top-down complexity modeling approach using longitudinal configuration data of a large-scale campus network. Our evaluations show that our model can accurately estimate configuration complexity metrics given only high-level design parameters. Discrepancies when present were mainly due to redundant configuration lines introduced by network operators. Our models provided important insights when applied to analyzing a major routing design change made by the operators undertaken with an explicit goal to lower design complexity. Our model indicated that while some of the design changes were useful in lowering complexity, others in fact were counter-productive and increased complexity. Further, our models helped point out alternate designs that could further lower complexity.

2 Dimensions of Routing Design

According to most computer networking textbooks, routing design is nothing more than selecting and configuring a single interior gateway protocol (IGP) such as OSPF on all routers and setting up one or more BGP routers to connect to the Internet. In reality, as one would quickly discover from meetings and online discussion forums of the operational community, network operators consistently rate routing design as one of the most challenging tasks.

In this section, using a toy example, we briefly break down the challenges of routing design along two structural dimensions, each made of a distinct logical building block. The goal is to identify the general sources of its complexity by exposing the major design choices that operators must make.

Consider Fig. 1, which illustrates a hypothetical company network that spans two office buildings. Assume that the physical topology has been constructed, including three subnets (Sales, Support and Data Center) in the main building and two additional subnets in building 2.

2.1 Policy groups

An integral part of almost every enterprise's security policy is to compartmentalize the flow of corporate information in its network. For the example network, there are two categories of users: Sales and Support. Suppose the Data Center subnet contains accounting servers that should be accessible only by the Sales personnel. A corresponding requirement of routing design would be to ensure that only the Sales subnets have good routes to reach the Data Center subnet.

We refer to the set of subnets belonging to one user category and have similar reachability requirements as *policy group*. We note that policy groups are similar to policy units introduced in [6], though there are some differences (see Sec. 9). A primary source of complexity for routing design is to support the fine grained reachability requirements of policy groups. This is particularly challenging since business stipulations often imply that subnets of a policy group may need to be distributed across multiple buildings, multiple enterprise branches, or even multiple continents.

The operator faces several choices in designing networks to meet these reachability requirements. The operator may choose to deploy a single IGP over the entire network to allow full reachability and then place packet filters on selected router interfaces to implement the required reachability policy. This is a viable solution for small networks. However, for medium to large networks, a large number of filtering rules need to be configured, and on many router interfaces. Doing so will likely introduce performance problems because packet filters incur per packet processing. In addition, according to a recent study [23], proper placement of packet filters in itself is a complex task, particularly when the solution must be resilient against link failures and other changes in the network topology.

Alternatively, the operator may choose to deploy a separate routing protocol instance to connect the subnets of each policy group. For the example network, two independent OSPF instances (OSPF 10 and OSPF 20) may be used to join the subnets of Sales and Support, respectively. Such a design is not straightforward either. First, the operator must decide which routers to include in each routing instance, subject to additional requirements such as resiliency. Second, the operator must select a small number of routers as border routers and configure connecting primitives [16, 19] to "glue" the different routing protocol instances together. (The next section has a detailed description of the tradeoffs involved in this step.) Last but not the least, the operator may need to configure route filters at the border routers to implement the required reachability policy. For the example network, route filters should be configured to prevent routes to Data Center subnets from leaking into Support.

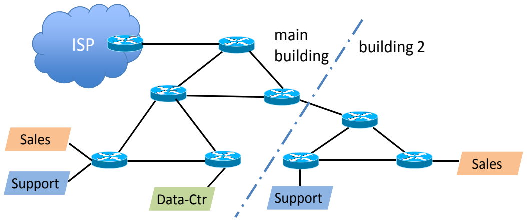


Figure 1: Example enterprise network spanning two office

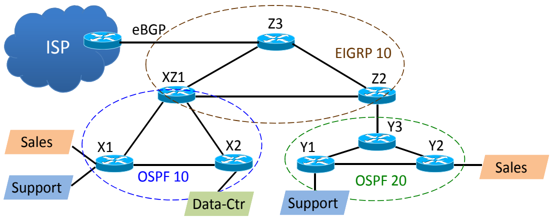


Figure 2: A design with multiple IGP instances

S_i	subnet i	R_i	router i
I_i	routing instance i	Z_i	policy group i
$\mathbf{M_C}$	connecting primitive matrix	$\mathbf{M_R}$	reachability matrix
$\mathbf{M_B}$	border-router matrix	$\mathbf{M_X}$	route-exchange matrix
$\mathbf{M_A}$	arc matrix	T_i	the internal routes that I_i has
W_i	the entire set of routes that I_i has	$f(x)$	complexity of configuring a route filter to allow a set of routes x
$\lvert x \rvert$	size of the set x	K_{rr}	complexity of configuring one-way route redistribution
K_{sr}	complexity of configuring one static route	K_{bgp}	complexity of configuring a BGP peering session on one border router
K_{obj}	complexity of configuring object tracking for one static route	$h(i,j)$	a binary function denoting whether filtering is needed
$\mathbf{M_P}$	eBGP-peering matrix	K_{dr}	complexity of configuring a default route

Table 1: Notation table

2.2 Routing instances

In several recent papers [4, 16, 20], researchers have reveale the common use of multiple routing protocol instances i one enterprise network. This is not surprising as the evc lution of an enterprise's computer network parallels that c its business. Networks of different routing designs are fuse at times of company mergers and acquisitions. Also, large enterprises and universities are usually made of a group of autonomous business units with quasi-independent IT staff; each unit often has a large degree of freedom in managing its part of the enterprise network, including the choice of which routing protocol to use.

We adopt from these prior works the definition of a routing protocol instance, or simply *routing instance*, to refer to a connected component of the network where all member routers run the same routing protocol, use matching protocol parameters, and are configured to exchange routes with each other. When dealing with a routing design with multiple routing instances, the operator must weigh different options of joining the different routing instances together while implementing the company's security policy.

To illustrate, suppose the operator of the example network of Fig. 1 has created a routing design with three routing instances, as shown in Fig. 2. Each office building has its own routing instance (OSPF 10 or OSPF 20) while the EIGRP 10 instance serves as the backbone of the network.

The operator has chosen to use BGP to connect to the Internet. It is straightforward to configure an external BGP (eBGP) peering session and mutual route redistribution (RR) between the EIGRP and BGP instances to allow routes to be exchanged between the enterprise network and its ISP. The complexity increases significantly, however, if the operator needs to configure route filters to reject certain incoming and outgoing routes. The problem would be compounded for multi-homed enterprise networks because of additional policy requirements such as the designation of primary and backup or load balancing.

For connecting the routing instances of OSPF 10 and the backbone, the operator has chosen to use a single border router (i.e., XZ1) that participates in both routing instances. (For brevity, we do not consider the resiliency requirement in this example.) This removes the need for BGP peering to advertise routes across the boundaries of the instances. However, the operator must still configure route redistribution and possibly route filters to allow the injection of routes between the two instances. Another complication is that multiple routing instances may simultaneously offer routes to the same destination at a border router. The operator must implement the correct order of preference between the instances, sometimes requiring an override of some protocols' default Administrative Distance (AD) [4, 18] values on a per border router basis.

For connecting the routing instances of OSPF 20 and the backbone, the operator has adopted a different approach. Two border routers (i.e., Y3 and Z2) are used as in the case of the ISP connection. However, instead of using a dynamic protocol, the operator has configured two sets of static routes, on the two border routers respectively, to achieve reachability across routing instances. This design incurs considerable amount of manual configuration on a per destination prefix basis. For example, on Y3, static routes are required for destination prefixes not only within EIGRP 10 but also within OSPF 10. Clearly, due to its static nature, the design may not bode well when the subnet prefixes frequently change or dynamic re-routing is required. However, it has one advantage over dynamic mechanisms like BGP or co-location of routing processes in one border router: the packet forwarding paths across routing instances are much easier to predict when hardcoded. In contrast, both BGP and RR can result in routing anomalies in ways that are difficult to identify from their configurations [18].

3 Abstractions for Modeling Routing Design

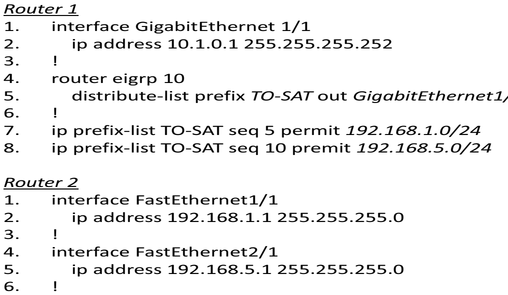
This section first presents a set of formal abstractions that capture the routing design primitives, and design objectives and constraints. These abstractions form the foundation for modeling design complexity. We then describe the metrics we used for quantifying the complexity of a given design instance that makes use of a specific set of design primitives and targets a specific network topology. Table 1 summarizes the notations used in this and the following sections.

3.1 Abstracting essential elements of routing design

We use $S_1, S_2, ...$ to denote the host subnets in the network. Each subnet is assigned a unique IP prefix address (e.g., "192.168.1.0/24"). We use $R_1, R_2, ...$ to denote the routers in the network. Each subnet connects to a router and uses that router as its gateway (i.e. the router routes all the packets generated by the subnet). A "route" is considered as an IP prefix address, plus additional attributes (e.g., weight) that may be used for calculation of a next-hop for the route. A router always has routes to all the connected subnets for which it is the default gateway. In addition, routes may be manually injected into a router via configuration of static routes (Sec. 5.3). Routers may exchange routes by running one or more dynamic routing protocols. To participate in each routing protocol, a router must run a separate routing process. Each routing process maintains a separate Routing Information Base, or RIB. The RIB contains all the routes known to the routing process, each associated with one or more next-hops. A router maintains a global RIB (also referred to as the forwarding information base, or FIB, in the literature), and uses selection logic to select routes from its routing process RIBs, as well as routes to connected subnets and statically configured routes, to enter the global RIB. A router uses the global RIB to make forwarding decisions. (Readers are referred to [20] for a more detailed description.)

We say that "a router R_i has a route to a subnet S_j", if the prefix address of S_j matches a route in the global RIB of R_i. Furthermore, we say that "a subnet S_i is routable from another subnet S_j", if the gateway router for S_j has a route to S_i. Finally, since this paper concerns only the routing design and uses routing as the only mechanism to implement reachability, we use the terms "reachable" and "routable" inter-changeably.

Let $\mathcal{I} = \{I_1, I_2, ...\}$ denote the set of routing instances (Sec. 2.2), an essential routing design component that abstracts route propagation in the network. As described in Sec. 2.2, routing processes in the same routing instance exchange all their routes freely. As a result, all the routing processes share the same set of routes. To change this behavior, route filters are typically used to filter route updates between routing processes (Sec. 4). On the other hand, routing processes in different routing instances do not exchange any route. To change this behavior, *connecting primitives* must be used (e.g., static routes, route redistribution and BGP). The routers where connecting primitives are implemented are termed *border routers*.



```
Router 1
interface GigabitEthernet 1/1
   ip address 10.1.0.1 255.255.255.252
!
router eigrp 10
   distribute-list prefix TO-SAT out GigabitEthernet1/1
!
ip prefix-list TO-SAT seq 5 permit 192.168.1.0/24
ip prefix-list TO-SAT seq 10 premit 192.168.5.0/24

Router 2
interface FastEthernet1/1
   ip address 192.168.1.1 255.255.255.0
!
interface FastEthernet2/1
   ip address 192.168.5.1 255.255.255.0
!
```

Figure 3: Configuration snippets of two routers.

We assume that we are given the set of routing instances $\mathcal{I}$ and their member routers and routing processes. We are also given the connecting primitive matrix $\mathbf{M_C}$. Each cell $\mathbf{M_C}(i,j)$ specifies the connecting primitive used by I_i and I_j, to allow routes to be sent from I_i to I_j.

In this paper, we focus on the primary use of routing design: implementing reachability policies. The primary layer-three mechanisms to implement reachability are the connecting primitives and route filters. We do not model the selection logic, which is used to prefer one routing path over another, as this is typically used for traffic engineering purposes, rather than implementing reachability.

3.2 Abstracting design objectives and constraints

The design objectives and constraints considered in this paper include reachability and resiliency, as well as routing path policies. First, to capture the reachability requirements, it is assumed that we are given the reachability matrix $\mathbf{M_R}$. Each cell $\mathbf{M_R}(i,j)$ denotes whether the subnet S_i can reach the subnet S_j. Note that in the routing design we only consider reachability at the subnet level. We do not consider host-level reachability as it is typically implemented by data plane mechanisms such as packet filters.

To capture the resiliency requirement, we assume that we are given the border-router matrix $\mathbf{M_B}$. Each cell $\mathbf{M_B}(i,j)$ specifies the set of I_i's border routers that enable I_i to advertise routes to I_j. Note that a routing instance may use different border routers to communicate with different neighboring instances.

To capture the path policies, it is assumed that we are given the route-exchange matrix $\mathbf{M_X}$. Each cell $\mathbf{M_X}(i,j)$ specifies the set of routes that I_i should advertise to I_j to meet the reachability requirement. We assume that the routes in the matrix is in the most aggregated form. Clearly the set of *external* routes that I_i has may be calculated as $\bigcup_j \mathbf{M_X}(j,i)$. Let T_i denote the set of *internal* routes that I_i has (i.e., routes originated by subnets inside I_i). Let W_i denote the entire set of routes that I_i has, which may be calculated as follows:

$$W_i = (\bigcup_j \mathbf{M_X}(j,i)) \bigcup T_i \tag{1}$$

3.3 Measuring complexity

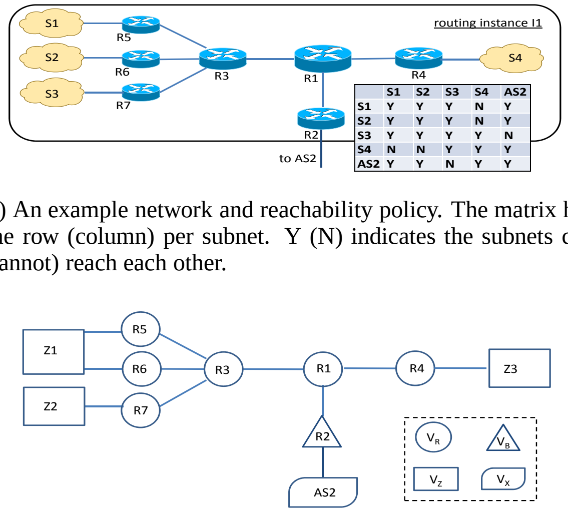
Using these abstractions, we are able to precisely define the objectives, or the correctness criteria, of a routing design, and reason about how a combination of routing primitives (e.g., routing instances, static routes, route filters, etc.) will meet the objectives. We then leverage metrics developed by previous work to measure how the choice of different routing primitives may impact the complexity of the resulting network.

The particular metric that we use is proposed by [5], which captures the complexity in configuring a network by counting the number of *reference links* in the device configuration files. Basically a reference link is created when a network object (e.g., a route filter, a subnet) is defined in one configuration block, and is subsequently referred to in another configuration block, in either the same configuration file or a different file. As an example, consider Fig. 3 that shows configuration snippets from two routers. The referential links are shown in italics. In line 5 in Router 1's configuration, a route filter named TO-SAT is applied to the interface GigabitEthernet1/1 to filter two routes in the outgoing direction. This line introduces two reference links: one to the name of filter (defined in line 7-8), and the other to the name of the interface (defined in line 1). Moreover, the definition of the route filter (lines 7-8) introduces two reference links to the two subnet prefixes, which are defined in Router 2's configuration file (line 2 and 5). Clearly, the existence of reference links increases the configuration complexity as it introduces dependencies between configuration blocks either in the same configuration file or in different configuration files.

We choose to use this metric because it has been extensively validated in [5] through operator interviews. Our own interaction with operators also suggests that the metric reflects operator perceived complexity reasonably well. We note that other complexity metrics have been proposed in [5] such as the number of routing instances, and the number of distinct router roles. Many of these other metrics are relatively straight-forward to estimate from the design. For example, the number of distinct router roles could be estimated based on the insight that border and non-border routers play different roles. Further, we have observed in our evaluation settings that the reference link metric shows the most variation across designs, making it particularly useful in facilitating comparisons.

4 Modeling Intra-Instance Complexity

This section presents a framework for estimating complexity existing *within* a routing instance. We first show that such complexity results from the need to install route filters inside the routing instance, in order to implement the different reachability requirements of different subnets. We then present models to quantify the complexity associated with such route filters. In doing so, our models determine the route filter placement and the filter rules.



	S1	S2	S3	S4	AS2
S1	Y	Y	Y	N	Y
S2	Y	Y	Y	N	Y
S3	Y	Y	Y	Y	N
S4	N	N	Y	Y	Y
AS2	Y	Y	N	Y	Y

(a) An example network and reachability policy. The matrix has one row (column) per subnet. Y (N) indicates the subnets can (cannot) reach each other.

(b) The per-instance routing graph of routing instance I1

Figure 4: Illustrating need for route filters.

4.1 Source of intra-instance complexity

The complexity within a routing instance primarily comes from the route filters installed inside the instance. By definition, all routing processes of the same routing instance have the same routing tables. This means that all the subnets connecting to those routing processes will have the same reachability toward other subnets. If this is not desired, route filters must be used to implement reachability policies inside a routing instance.

As an example, consider the example network shown in Fig. 4a. Routers R1-R7 and subnets S1-S4 are placed in routing instance I1. Border router R2 runs eBGP with another autonomous system AS2 and injects eBGP learned routes to I1. The figure also shows the desired reachability matrix. To implement the reachability matrix, route filters must be carefully placed. For example, to prevent S1 and S2 from reaching S4, while permitting S3 to reach S4, route filters must be installed between R3 and R5, and between R3 and R6. Similarly, a route filter must be installed between R1 and R4 to prevent S4 from reaching S1 and S2. In addition, another route filter must be installed between R3 and R7, to prevent S3 from reaching the external routes of I2.

In general, the degree of diversity in terms of reachability among subnets of the same routing instance directly impacts the amount of filtering required, which in turn determines the complexity inside that instance. To capture this degree of diversity, we leverage the notion of *policy groups* discussed in Sec 2.1.

Policy groups: Formally, let $\mathcal{Z} = \{Z_1, Z_2, ...\}$ denote the set of policy groups in a network. A policy group $Z_i \in \mathcal{Z}$ is a set of subnets that (i) can reach each other, and (ii) are subject to the same reachability treatment *toward* other subnets (e.g., if a subnet $s_a \in Z_i$ can reach another subnet

$s_b \in Z_j$, then all subnets in Z_i must be able to reach s_b as well). Clearly, policy groups divide the set of all subnets, and each subnet belongs to one and only one policy group. The set of policy groups of a given network can be easily derived from the reachability matrix $\mathbf{M_R}$. In the example in Fig. 4a, S1 and S2 forms a policy group, while S3 and S4 each constitute a separate policy group.

By definition, there is no need for filtering within a policy group. Thus if a routing instance contains only a single policy group, the intra-instance complexity is zero. On the other hand, if a routing instance contains subnets of multiple policy groups, route filters must be installed among them to implement their different reachability constraints, and thus incur complexity.

4.2 Modeling the complexity

Intuitively, the degree of complexity of a given routing instance I_a depends on two factors:

- The number of route filters installed inside I_a, as each installation of a filter creates a reference link to the name of that filter (e.g., line 5 of router 1 in Fig 3).
- The complexity associated with each filter, which is measured by the number of rules in each filter, as each rule creates a reference link to a prefix address (e.g., lines 7-8 of router 1 in Fig 3).

Below we model the two factors separately.

4.2.1 Estimating number of filters

In order to estimate the number of route filters needed to be installed inside a given routing instance I_a, we first introduce an undirected graph $G_a(V_R, V_Z, V_X, V_B, E)$, called the *per-instance routing graph* of I_a. We then show how we use this graph to do the estimation for different network topologies.

Per-instance routing graph: The purpose of the per-instance routing graph is to model how policy groups are inter-connected. There are four types of nodes in the graph for a given routing design: V_R, V_Z, V_X and V_B. V_R denotes the set of routers that participate in this routing instance. V_Z denotes the set of policy groups that are placed inside this routing instance. V_X denotes networks external to this routing instance (i.e., other routing instances in the same AS and external ASes as well), whose routes are injected into this routing instance by one or more border routers. Finally, V_B denotes the set of border routers of this routing instance.

E denotes the set of edges. First, there is an edge between two nodes $v_i, v_j \in V_R$ if the two routers are physically connected, and the corresponding routing processes running on them are *adjacent*, i.e., can exchange routing updates [20]. Second, there is an edge between $v_i \in V_Z$ and $v_j \in V_R$ if one or more subnets in policy group v_i connect to the routing process running on v_j. Finally, there is an edge between $v_i \in V_X$ and $v_j \in V_B$, if the border router v_j injects the routes of the external network v_i.

For example, the per-instance routing graph of I_1 in Fig. 4a is shown in Fig. 4b.

Determine filters needed for one policy group: Using the per-instance routing graph, we determine the route filters needed for implementing the reachability of a policy group $v_i \in V_Z$ *toward* other subnets. First, consider every policy group $v_j \in V_Z$. If v_j contains one or more subnets that v_i can not reach, then a route filter must be placed on every possible path between v_i and v_j on the per-instance routing graph, to filter out routing updates corresponding to those subnets before they reach any gateway router of v_i. Similarly, consider every external network $v_k \in V_X$. If there exist one or more subnets in v_k that v_i cannot reach, a route filter must be placed on every possible path between v_i and v_k to filter routing updates corresponding to those subnets as well.

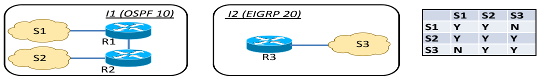
Upper and lower bounds on the number of filters: In both cases described above, the *upper bound* on the number of route filters needed for policy group v_i is the total number of paths between v_i and v_j (v_k), summed over all v_j and v_k for which filtering is needed. The upper bound can always be achieved by placing the filters on the on gateway routers of v_i. The *lower bound* is the number of links in the smallest edge-cut set between v_i and v_j (v_k), summed over all v_j and v_k for which filtering is needed. However, the lower bound may not always be achievable, as some links may be included in the smallest edge-cut sets between multiple pairs of policy groups. For example, in Fig. 4b routing updates of Z_3 must be filtered before they reach Z_1, as Z_1 is not allowed to reach Z_3. While one smallest edge-cut set between Z_1 and Z_3 is the link $R_1 - R_3$, we cannot place the filter on that link, as doing so would wrongfully prevent Z_2 from getting those routing updates.

The lower bound can be achieved for a special type of star topology, which we believe is typical in many enterprise networks. In this type of topology, any path between a pair of policy groups, or between a policy group and an external network, always goes through the core router. This ensures that the paths between the core tier and different policy groups do not share any common router. Given this special topology, it may be shown that (i) the core-tier will have the complete set of routes, and (ii) it is sufficient to place the route filters between the core tier and each policy group. Hence it is now feasible to place the filters on the smallest edge-cut set between the core tier and each policy group.

4.2.2 Estimating number of rules in each filter

Consider using a route filter to implement a policy group v_i's reachability constraint toward another policy group v_j. The number of rules in this filter depends on the number of routes to be blocked from v_j to v_i, as one route translates to one filter rule (see Fig. 3 for an illustration).

For example, as we have discussed above, a route filter must be installed between Z1 and Z3 in the toy network (Fig. 4b), to prevent the routes of S1 and S2 from being advertised to S4. The number of rules in this filter will be two, as there are two prefixes to be blocked. (Note that the



	S1	S2	S3
S1	Y	Y	N
S2	Y	Y	Y
S3	N	Y	Y

Figure 5: A toy network with two routing instances.

number of rules may be reduced, if several prefixes can be aggregated into a larger prefix. For simplicity we do not consider such route aggregation in this work.)

5 Modeling Inter-Instance Complexity

This section presents a framework for estimating the inter-instance complexity. We show that this complexity results from the use of connecting primitives. We then present models for estimating the complexity for the three typical connecting primitives described in Sec. 2.2: route redistribution, static and default routes, and BGP.

5.1 Source of inter-instance complexity

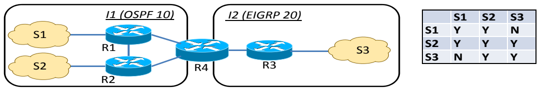
The inter-instance complexity comes from the need for connecting primitives to connect multiple routing instances. Consider the toy network shown in Fig. 5 as an example. There are two routing instances: I1 running OSPF with process ID 10, and I2 running EIGRP with process ID 20. I1 contains subnets S1 and S2, and I2 contains S3. The reachability policy specifies that S1 and S2, as well as S2 and S3 can reach each other, but S1 and S3 can not. Given the network as such, I1 and I2 cannot exchange any route, and thus cannot communicate. To change this behavior and implement the reachability between S1 and S3, one or more border routers must be deployed to physically connect the two routing instances, and in addition, a connecting primitive must be configured on each border router to enable route exchange.

An important factor that impacts the degree of inter-instance complexity is the resiliency requirement (Sec. 3.1), which specifies the number of border routers (Sec. 3.2) each routing instance should have. While having more border routers improves the resiliency of the design, it also introduces potential anomalies (e.g., routing loops) and complicates the configuration, as we will show in the next section.

In this section we focus on the basic scenario with a single border router for each routing instance (i.e., minimum resiliency). We discuss the case with multiple border routers in the next section.

5.2 Route redistribution

The first connecting primitive we consider is route redistribution, which dynamically sends routes from one routing instance to another. Using route redistribution to connect two routing instances requires having a common border router that runs routing processes in both routing instances. The border router then is configured to redistribute routes from one routing instance to the other, and vice versa. (Note that route redistribution must be separately configured for each direction.) For example, Fig. 6a illustrates the design using route redistribution for the network shown in Fig. 5.



	S1	S2	S3
S1	Y	Y	N
S2	Y	Y	Y
S3	N	Y	Y

(a) The network design using route redistribution.

```
Router 4
1.    router ospf 10
2.       redistribute eigrp 20
3.    !
4.    router eigrp 20
5.       redistribute ospf 10 route-map OSPF-TO-EIGRP
6.    !
7.    route-map OSPF-TO-EIGRP permit 10
8.       match ip address 1
9.    !
10.   access-list 1 permit S2
```

(b) Configuration snippet of the border router R4

Figure 6: Design using route redistribution for the network shown in Fig. 5.

Router R4 is the border router and is configured to redistribute routes between I1 and I2.

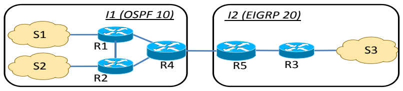
Fig. 6b shows the relevant configuration snippet of R3 in Cisco IOS syntax, with referential links highlighted in italics. Line 1 and 4 create two routing processes, one participate in each routing instance. Line 2 and 5 redistribute routes from I1 to I2 and from I2 to I1 respectively.

We note that by default, route redistribution will redistribute *all* the active routes from one instance to the other [17]. For example, in Fig. 6a, R4 will redistribute routes of both S1 and S2 to I2. This enables S3 to reach both S1 and S2, which does not conform to the reachability policy as shown. To change the default behavior, a route filter (in the form of a route-map) must be used in conjunction with route redistribution, as shown in line 5 in Fig. 6b. Such a filter permits a subset of routes specified by the filtering rules (line 8 and 10), and blocks the rest routes.

Modeling complexity: Consider route redistribution from I_i to I_j. Route redistribution in the other direction may be modeled similarly and separately. As shown above, the configuration may include two components: (i) configuration of the route redistribution itself, and (ii) configuration of the route filter, which is required if only a subset of I_j's routes should be redistributed. Let K_{rr} denote the complexity of configuring the route redistribution itself. Let the function $f(x)$ denote the complexity of configuring and installing a route filter with x rules (i.e., the filter allows x routes). We note that $f(x)$ includes (i) the complexity of configuring the filtering rules, which is linear to the number of rules, and (ii) the complexity of installing the filter by referring to its name, which is a constant factor. In addition, we let $h(i,j)$ be the following binary function that denotes whether a filter is needed: (Recall that a filter is not needed if all the routes I_i has, i.e. W_i, can be redistributed into I_j.)

$$h(i,j) = 0, \text{ if } \mathbf{M_X}(i,j) = W_i; \tag{2}$$

$$h(i,j) = 1, \text{ otherwise.} \tag{3}$$



	S1	S2	S3
S1	Y	Y	N
S2	Y	Y	Y
S3	N	Y	Y

(a) The network design using either static routes or BGP.

```
Router 4
1.    router ospf 10
2.        redistribute static
3.    !
4.    ip route S3 R5

Router 5
5.    router eigrp 20
6.        redistribute static
7.    !
8.    ip route S2 R4
```

(b) Configuration snippets of the border routers using static routes.

Figure 7: Design using static routes for the network shown in Fig. 5.

The overall complexity denoted by $C_{\text{rr}}(i,j)$ can be calculated as follows:

$$C_{\text{rr}}(i,j) = K_{\text{rr}} + f(\mathbf{M_X}(i,j)) * h(i,j) \tag{4}$$

5.3 Static routes

Another way to connect the two routing instances in Fig. 5 is to use static routes, which can be viewed as manually entered routing table entries. This design is shown in Fig. 7a. Each routing instance now must have its own border router (R4 and R5) that participates in only that routing instance. Static routes are configured on the border routers and point to destination subnets in the other routing instance. One static route is needed for every destination subnet. Further, the static routes are redistributed into the respective routing instance so that internal routers in the routing instance also have those routes.

Fig. 7b shows the relevant configuration snippets of the two border routers, with referential links highlighted in italics. On R4, a static route is configured and installed in line 4. The static route points to S3 as the destination, and specifies R5 as the next-hop to reach the destination. Having this static route installed, R4 now has a route to S3. Further, line 2 redistributes any static route configured on the router into I1, so that other routers of I1 (i.e., R1 and R2) also have a route to reach S3. Similarly, a static route to S2 is configured on R5 (line 8) and redistributed (line 6) to I2.

Modeling complexity: Consider using static routes to allow I_j to reach a set of subnets $\mathbf{M_X}(i,j)$ in I_i. Let $|\mathbf{M_X}(i,j)|$ denote the size of $\mathbf{M_X}(i,j)$. Since one static route is needed for each route in $\mathbf{M_X}(i,j)$, there will be $|\mathbf{M_X}(i,j)|$ static routes to configure. Let K_{sr} denote the complexity of configuring a single static route. The total complexity denoted by $C_{\text{sr}}(i,j)$ can be calculated as follows:

$$C_{\text{sr}}(i,j) = |\mathbf{M_X}(i,j)| * K_{\text{sr}} \tag{5}$$

```
Router 4
1.     router ospf 10
2.        redistribute bgp 64501
3.        !
4.     router bgp 64501
5.        neighbor R5 remote-as 64502
6.        neighbor R5 distribute-list 1 out
7.        redistribute ospf 10
8.     !
9.     access-list 1 permit S2

Router 5
10.    router eigrp 20
11.       redistribute bgp 64502
12.    !
13.    router bgp 64502
14.       neighbor R4 remote-as 64501
15.       redistribute eigrp 20
16.    !
```

Figure 8: Configuration snippets of the border routers using BGP, for the network shown in Fig. 7a

Default routes: A *default route* is a special case of static routes, which injects a default gateway to the router it is configured on. A default route has a constant complexity (denoted as K_{dr}). For example, in Cisco IOS syntax the command to configure a default route is "ip route 0.0.0.0 0.0.0.0 *next-hop-IP* ". This is essentially the same command for configuring static routes, except that the destination prefix takes the special form "0.0.0.0 0.0.0.0", which will match any IP address. Clearly the complexity of this command is one, as it creates a single referential link to the address of the next-hop router.

5.4 BGP

A third connecting primitive is BGP, which is a dynamic routing protocol that allows routes to be exchanged among routing instances. BGP typically requires each routing instance to have its own border router. The design using BGP for the same example network is shown in Fig. 7a. Again R4 and R5 are the border routers for I1 and I2 respectively. In addition to running the respective IGP routing process, R4 and R5 each also runs a separate BGP routing process. A BGP peering relationship is established between R4 and R5, so that R4 can advertise S2 to R5, and R5 can advertise S3 to R4. R4 and R5 also redistribute the learned BGP routes to their respective routing instance, so that other routers in the routing instance have those routes too.

Fig. 8 shows the relevant configuration snippets of R4 and R5. Configuring R4 involves: (i) starting a BGP routing process (line 4); (ii) redistributing routes from the IGP into the BGP process (line 7); (iii) establishing a BGP peering session with the neighboring border router (R5), and exchanging routes with it (line 5); (iv) installing an optional route filter to restrict the routes to be advertised (line 6); and (v) redistributing the learned BGP routes into IGP (line 2). Similar configuration is done on R5 too. We wish to note two things here. First, the BGP process does not have any route by default, and hence routes must be explicitly redistributed from the IGP to the BGP, i.e., the step (ii) above. Second,

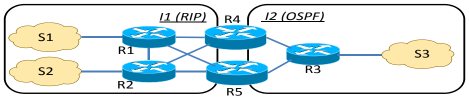


Figure 9: Both border routers R4 and R5 are performing mutual route redistribution between I1 and I2. Assume full reachability among all subnets.

BGP advertises all its routes to neighbors by default. If this is not desired, a route filter must be used to restrict routes to be advertised, i.e. the step (iv) above.

Modeling complexity: Consider that I_i advertises a set of routes to I_j using BGP. The complexity of configuring BGP on I_i's border router consists of three components: (i) the complexity of configuring the BGP session itself, including configuring the BGP process and the peering relationship with a neighbor; (ii) the complexity of configuring two-way route redistribution between the IGP and the BGP processes; and (iii) the complexity of configuring a route filter, if it is needed (i.e., only a subset of I_i's routes can be advertised). Let K_{bgp} denote the complexity of configuring the BGP protocol itself. Let $f(x)$ and $h(i,j)$ be the same functions as defined in Sec. 5.2. The total complexity denoted by $C_{\text{bgp}}(i,j)$ can be calculated as follows:

$$C_{\text{bgp}}(i,j) = K_{\text{bgp}} + 2 * K_{\text{rr}} + f(\mathbf{M_X}(i,j)) * h(i,j) \tag{6}$$

6 Complexity With Multiple Border Routers

We now consider designs where a routing instance uses multiple border routers to connect to another routing instance. An example of such a design is shown in Fig. 9, where both R4 and R5 are configured to perform two-way route redistribution between I1 and I2. The main benefit of using multiple border routers is increased resiliency, i.e., even if one border router in Fig. 9 fails, I1 and I2 can still communicate through the other one. On the other hand, using multiple border routers can cause anomalies such as routing loops. To prevent the anomalies, additional configuration will be required, which may introduce more referential links and increase complexity.

In this section, we model the additional complexity resulting from both the *safety* and *resiliency* requirements with multiple border routers:

• **Safety**: the routing must function correctly when all the border routers are alive and running, e.g., no routing loop will occur;

• **Resiliency**: when one or more border router and/or link is down, the routing must be able to adapt and re-route traffic though live routers and/or links.

We examined all three connecting primitives and found that (i) for route redistribution, additional mechanisms are required to ensure safety; (ii) for static routes, additional mechanisms are required to ensure resiliency; and (iii) for BGP, no additional mechanism is needed. Below we model the complexity of each of these connecting primitives.

6.1 Ensuring safety with route redistribution

Consider a design scenario where route redistribution is configured on multiple border routers to redistribute routes from I_i to I_j. The complexity of this design depends on what connecting primitive is used to send routes in the reverse direction (i.e., from I_j to I_i), as we discuss below.

On the one hand, if no connecting primitive or a different connecting primitive than route redistribution is used in the reverse direction, the complexity C_{rr} is simply the single border-router complexity (Sec. 5.2) multiplied by the number of border routers, i.e.,

$$C_{\text{rr}}(i,j) = (K_{\text{rr}} + f(\mathbf{M_x}(i,j)) * h(i,j)) * |\mathbf{M_B}(i,j)| \tag{7}$$

Recall that $\mathbf{M_B}(i,j)$ denotes the set of border routers that I_i uses to reach I_j, which is an input to our framework (Sec. 3.2). $|\mathbf{M_B}(i,j)|$ denotes the size of this set, i.e., the number of border routers.

On the other hand, if route redistribution is also used in the reverse direction, then a potential anomaly called *route feedback* may occur. Route feedback happens when a route is first redistributed from I_i to I_j by one border router, but then is redistributed back from I_j to I_i by another border router. For example, in the network in Fig. 9, S1 may be first redistributed from I1 (RIP) to I2 (OSPF) by router R4. So router R5 learns the route from both RIP and OSPF. If R5 prefers the OSPF-learned route, it will redistribute the route back to RIP. Route feedback *can* lead to several problems such as routing loops and route oscillations [17]. Clearly route feedback can happen only when mutual route redistribution is conducted by multiple border routers between two routing instances.

As a common conservative solution to this issue, a route filter is used in conjunction with route redistribution to prevent any route from re-entering a routing instance where it originally came. In the above example, a route filter should be installed on R4 and R5 to allow only the route S3 to enter I1, and prevent routes of S1 and S2 from re-entering I1. Note that such a filter may be already in place to implement reachability as described in Sec. 5.2 (i.e., to only allow a subset of routes of I_j to be sent to I_i, and block all other routes). In such case, there is no additional complexity introduced. Only in the case where the filter is not needed otherwise (i.e. $\mathbf{M_X}(j,i) = W_j$), a route filter needs to be configured for the sole purpose of preventing route feedback. To summarize, in the mutual route redistribution case, the total inter-instance complexity of using route redistribution to send routes from I_i to I_j is:

$$C_{\text{rr}}(i,j) = (K_{\text{rr}} + f(\mathbf{M_x}(i,j))) * |\mathbf{M_B}(i,j)| \tag{8}$$

The complexity on the reverse direction can be similarly modeled.

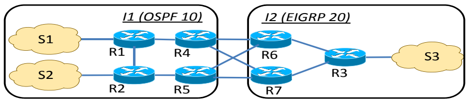


Figure 10: Static routes are configured on all border routers R4 - R7. Assuming full reachability among all subnets.

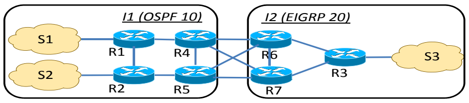


Figure 11: BGP is configured on all border routers R4 - R7. Assuming full reachability among all subnets.

6.2 Ensuring resiliency with static routes

Consider a design scenario where a routing instance I_i uses static routes to reach a set of destination subnets in I_j, as is the case for routing instances I1 and I2 in the example network in Fig. 10. On each border router of I_i (e.g., R4 in Fig. 10), and for each destination subnet in I_j (e.g., S3), multiple static routes may be defined, each using a different border router of I_j as the next-hop. For example, two static routes may be configured on R4 to reach S3, one using R6 as the next-hop, and the other using R7. We assume that we are also given as input an *arc matrix* $\mathbf{M_A}$, where each cell $\mathbf{M_A}(i,j)$ specifies the set of arcs from the set of border routers in I_i to the set of border routers in I_j. An "arc" is said to exist from one border router $R_a \in \mathbf{M_B}(i,j)$ to another border router $R_b \in \mathbf{M_B}(j,i)$, if there exists a static route on R_a that uses R_B as the next hop.

One limitation with static routes is that they may not be able to automatically detect the failure of the next-hop router or the link in between, and will continue to try to route traffic to the bad path, even when other valid paths exist. This will result in packets being dropped. For example, In Fig. 10, when there is no failure, R4 will load balance the two static routes and use both R6 and R7 to route traffic to I2. R5 will do the same thing. However, if R6 fails, R4 and R5 will not be able to detect the failure or cancel the corresponding static route that uses R6 as the next-hop. Instead, they will continue to try to route half of the traffic to R6, resulting in those packets being dropped.

A common solution to this problem is using *object tracking* [10] along with each static route. In doing so, each static route involves referring to an object tracking module. At a high level, object tracking will periodically ping the destination subnet of the static route, using the same next-hop router as specified in the static route. When a failure occurs and the destination is no longer reachable via the particular next-hop, the static route will be removed from the RIB at that point.

Let K_{obj} denote the complexity of installing object tracking to one static route. The total complexity of using static routes to enable I_j to reach I_i can be modeled as follows, assuming each arc contains static routes to reach all subnets in $\mathbf{M_X}(i,j)$:

$$C_{\text{sr}}(i,j) = |\mathbf{M_A}(i,j)| * |\mathbf{M_x}(i,j)| * (K_{\text{sr}} + K_{\text{obj}}) \quad (9)$$

That is, the total complexity is the single arc complexity (which includes both the complexity of configuring the set of static routes, and the complexity of installing object tracking to each static route), multiplied by the total number of arcs from I_i to I_j (denoted by $|\mathbf{M_A}(i,j)|$).

6.3 The case with BGP

Consider a design scenario where BGP is used to enable two routing instances I_i and I_j to exchange routes. An example of such design is shown in Fig. 11. Each border router runs *eBGP* peering sessions with one or more border routers of the other routing instance, and runs *iBGP* peering sessions with each border router of the same routing instance. For example, R4 in Fig. 11 runs eBGP peering sessions with R6 and R7, and an iBGP peering session with R5.

We assume that we are also given as input an *eBGP-peering matrix* $\mathbf{M_P}$, where each cell $\mathbf{M_P}(i,j)$ specifies the set of eBGP peering sessions between border routers of I_i and I_j. For the example network shown in Fig. 11, $\mathbf{M_P}(1,2)$ is 4. In addition, we assume that each border router runs an iBGP session with every other border router in the same routing instance, which is required for iBGP to work correctly. Hence each border router of I_i runs $(|\mathbf{M_B}(i,j)|-1)$ iBGP sessions.

The complexity of configuring I_i's border routers consists of three parts: (i) the complexity of configuring the eBGP sessions with border routers of I_j, which includes configuring route filters if needed to restrict routes to be advertised; (ii) the complexity of configuring the iBGP sessions among border routers of I_i; and (iii) the complexity of configuring the route redistribution between the BGP process and the IGP process. Hence the total complexity may be calculated as follows:

$$\begin{aligned} C_{\text{bgp}}(i,j) = {} & (K_{\text{bgp}} + f(\mathbf{M_X}(i,j)) * h(i,j)) * |\mathbf{M_P}(i,j)| + \\ & K_{\text{bgp}} * |\mathbf{M_B}(i,j)| * (|\mathbf{M_B}(i,j)| - 1) + \\ & 2 * K_{\text{rr}} * |\mathbf{M_B}(i,j)| \qquad (10) \end{aligned}$$

7 Evaluation

In this section, we evaluate our framework using configuration files of a campus network of a large U.S. university with tens of thousands of users. Our data-set includes multiple snapshots of the configuration files of all switches and routers from 2009 to 2011. It also includes multiple snapshots of the complete layer-two topology data, each collected using Cisco CDP at the same time each configuration snapshot was collected. The network has more than 100 routers and more than 1000 switches, all of which are Cisco devices. It also has tens of thousands of user hosts, and around 700 subnets, most of which are /24.

K_{rr}	K_{sr}	K_{dr}	K_{bgp}	K_{obj}	$f(x)$
1	2	1	2	1	$\|x\| + 2$

Table 2: Realizing framework parameters

	DATA	RSRCH	GRID	INT
DATA	-	*H-1*	×	✓
RSRCH	✓	-	✓	✓
GRID	×	*H-1-1*	-	×
INT	*D-1*	*H-1-1*	×	-

Table 3: Each cell (*row*, *column*) shows whether the policy group *column* can be reached by the policy group *row*. ✓/× means full/no reachability. *D-1*, *H-1* and *H-1-1* each denotes a subset of the subnets in *DATA* and *RSRCH*, which can be reached by the corresponding *row*. *H-1-1* is in turn a subset of *H-1*.

7.1 Framework validation

We first evaluate the accuracy of our framework in estimating complexity. In doing so, we run the framework on one of the configuration snapshots, and compare the predicted complexity numbers with the actual numbers obtained from measuring the configuration files directly.

7.1.1 Inferring model parameters and framework inputs

We only need to calculating the model parameters for the Cisco IOS platform, as this platform is exclusively used by the campus network. Obtaining these parameters is straightforward, as we just need to run the heuristics proposed in [5] on corresponding configuration blocks that relate to each parameter, and count the number of reference links introduced. The results are shown in Table 2.

To infer the inputs as described in Sec. 3.2, we used a methodology that combines reverse-engineering the configuration files and discussions with operators. We were able to identify the inputs as follows. Table 3 shows the policy groups and the reachability policies among them. Fig. 12a shows the topology and what policy groups each routing instance contains. In particular the campus network has two routing instances denoted as EIGRP and OSPF. There are also two policy groups in the network denoted as *DATA* and *RSRCH*. In addition, two external AS-es (denoted as GRID and INT) peer with this campus network. Each external AS can be viewed both as a single policy group and as a single routing instance. Finally, the $\mathbf{M_X}$ matrix, i.e., the set of routes exchanged between every pair of routing instances, is shown in Table 4.

7.1.2 Estimating intra-instance complexity

First, according to our framework, only the EIGRP instance will incur intra-instance route filters as it is the only instance that contains multiple policy groups.

Second, the EIGRP instance employs the typical star topology (Sec. 4.2.1), as shown in Fig. 12b. The border router R_1 also serves as the core router and connects the two policy groups: *DATA* and *RSRCH*. R_1 also directly connects to the other borders R_2, R_3 and R_4. Note that there is no direct link between the two policy groups, or between either policy group and R_2/R_3/R_4.

	EIGRP	OSPF	GRID	INT
EIGRP	-	all	*H-1-1*	*D-1*, *H-1-1*
OSPF	all	-	-	-
GRID	all	-	-	-
INT	all	-	-	-

Table 4: Each cell (*row*, *column*) shows the set of routes that routing instance *row* should advertise to routing instance *column*. "All" means that *row* should advertise all its routes (both internal and external ones) to *column*.

	EIGRP	OSPF	GRID	INT
EIGRP	7	1	6	30
OSPF	1	0	-	-
GRID	1	0	-	-
INT	2	-	-	-

Table 5: Estimated complexity for the original design. Each non-diagonal cell (*row*, *column*) shows the inter-instance complexity of advertising routes from *row* to *column*. The cells on the diagonal show the intra-instance complexity. "-" indicates that the two instances are not directly connected.

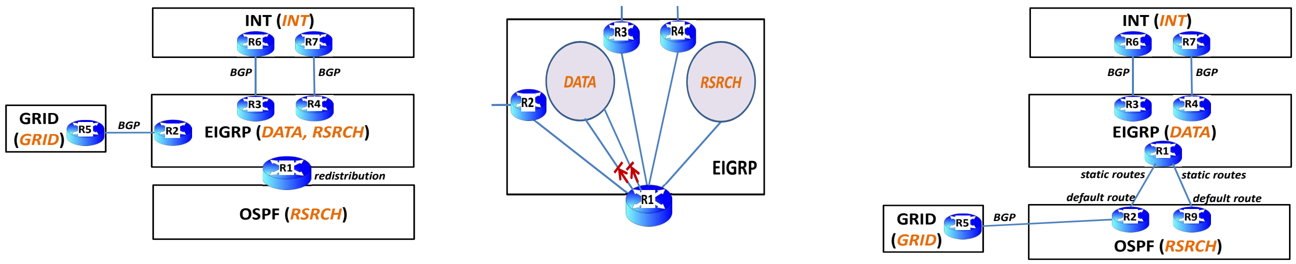
From the reachability matrix (Table 3), it is easy to see that intra-instance filtering is needed between the core router R_1 and the *DATA* policy group as only a subset of routes from R_1 can be sent to *DATA*, and in particular, the routes learned from GRID cannot be exposed to *DATA*. Using the model presented in Sec. 4, the route filter placement is determined and shown in Fig. 12b. Route filtering is not needed between the core router R_1 and *RSRCH*, because *RSRCH* can reach every other policy group. The predicted complexity is shown by the diagonal cells in Table 5.

Comparing with the actual configuration: We measured the actual configuration complexity in the configuration files. The result in shown in the diagonal cells in Table 6. As predicted, only the EIGRP routing instance incurs intra-instance route filters, and the filter placement is exactly as predicted. Furthermore, the measured complexity numbers also match the estimated value well.

7.1.3 Estimating inter-instance complexity

Using the models presented in Sec. 5, we estimate the inter-instance complexity, and show results in Table 5.

Comparing with the actual configuration: We compare the predicted inter-instance complexity with the complexity measured in the configuration files. The differences are shown in Table 6. We see that the majority of the predicted numbers match well with the actual configuration. There is a mismatch in the case of filtering routes between GRID and EIGRP. The measured value is greater than the prediction, which makes sense as the prediction is the minimum necessary complexity. The actual configuration may incur higher



(a) Instance-level topology of the original design.

(b) Detailed topology of EIGRP in the original design.

(c) Instance-level topology of the new design.

Figure 12: The original and new routing designs.

	EIGRP	OSPF	GRID	INT
EIGRP	$\epsilon = 0$	$\epsilon = 0$	$\epsilon = -6$	$\epsilon = 0$
OSPF	$\epsilon = 0$	$\epsilon = 0$	-	-
GRID	$\epsilon = -3$	$\epsilon = 0$	-	-
INT	$\epsilon = 0$	-	-	-

Table 6: Difference between complexity estimated using our models and the actual complexity measured from the configuration files for the original design.

	EIGRP	OSPF	GRID	INT
EIGRP	$\delta = -7$	$\delta = 7$	$\delta = -6$	$\delta = 0$
OSPF	$\delta = 29$	$\delta = 0$	$\delta = 6$	-
GRID	$\delta = -1$	$\delta = 1$	-	-
INT	$\delta = 0$	-	-	-

Table 7: Increase in the intra- and inter-instance complexity after the redesign.

complexity, for example, due to redundant configurations or suboptimal configurations.

In particular, the outgoing routes from EIGRP to GRID are subject to filtering as only a subset of EIGRP routes can be sent to GRID. We note that the filtering may be configured either at the redistribution point (i.e. permitting only the subset of routes to enter BGP), or within the BGP session (i.e. permitting only the subset of routes to be advertised to GRID). However, in the actual configuration, the exact same filtering is implemented at *both* places. This is redundant configuration, and results in unnecessary increase in the complexity. Further, GRID can advertise all its routes to EIGRP, so there is no route filter needed. However, in the actual configuration, an unnecessary filter is configured, which simply allows all routes to pass. As a result, three extra reference links were created.

Overall, these results confirm that our framework can accurately estimate the complexity of a given routing design.

7.2 Case study of a routing design change

The campus network experienced a major design change recently. The change was primarily motivated by the need to increase the resiliency of the original design. Thus as the second part of the evaluation, we apply our framework to compare the new routing design with the original one. We first use our framework to analyze the change in complexity due to the redesign. We then consider whether alternative designs could have met the same resiliency objectives but with lower complexity.

7.2.1 Impact of redesign on complexity

Fig. 12c illustrates the new instance-level graph after the network redesign was completed. The primary purpose of the redesign was to increase resiliency. In particular, the number of border routers connecting the OSPF instance to EIGRP was increased to two. In addition, two other changes were made: (i) the connecting primitive between EIGRP and OSPF was changed from route redistribution to static routes (configured on the EIGRP side) and default routes (configured on the OSPF side); and (ii) the subnets of the policy group *RSRCH* that were in the EIGRP instance were moved to OSPF. As a result, in the new design, EIGRP only contains subnets of the policy group *DATA*, while OSPF contains all subnets of the policy group *RSRCH*. Finally, we note that the policy groups and the reachability matrix were unchanged after the redesign.

Table 7 presents the change in complexity estimated by our framework. Overall, the total complexity in the new design increased. This is in part due to the fact that the resilience of the new design also increased, i.e., it used two border routers for the OSPF routing instance, compared to one in the old design. We note that the new design eliminated the intra-instance complexity in the EIGRP routing instance, as now EIGRP only contained a single policy group. On the other hand, the inter-instance complexity between EIGRP and OSPF increased in the new design, caused by the need to implement the different reachability requirements for the two policy groups *RSRCH* and *DATA*.

7.2.2 Could alternative designs lower complexity?

In the previous section, we noted that while the primary goal of the redesign was to improve resiliency, operators made two additional changes that were not strictly necessary to achieve this goal: (i) changing the connecting primitive between OSPF and EIGRP from redistribution to static/default routes; and (ii) moving all subnets of the RSRCH policy

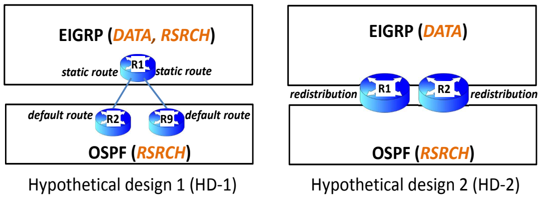


Figure 13: The two hypothetical designs.

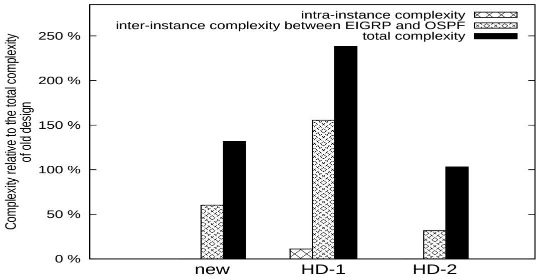


Figure 14: Comparison of complexity of different designs.

group to OSPF. We hypothesized these changes may have been made to lower complexity. To isolate the impact of each of these changes, we considered two hypothetical designs termed *HD-1* and *HD-2*, as shown in Fig. 13. Both designs use two border routers for OSPF, to achieve the same resiliency requirement as the new design. *HD-1* uses static and default routes to connect EIGRP and OSPF, and represents a design where only the first of the two additional changes above were made. *HD-2* involves a rearrangement of policy groups and represents a design where only the second of the two additional changes above were made. Route redistribution is used to connect the instances.

We apply our framework to estimate the complexity for both hypothetical designs. The results are shown in Fig 14. For ease of comparison, we normalized all bars to the total complexity of the original campus design. We see that while *HD-1* is a worse alternative design as its total complexity (third bar) increases compared to the actual new design, *HD-2* is a better alternative as its total complexity decreases compared to the actual new design.

We next seek to better understand why *HD-1* has higher complexity than the actual new design. The main difference between the two designs is whether the policy group $RSRCH$ is placed entirely in the OSPF routing instance (new design), or split across both OSPF and EIGRP (*HD-1*). We observe that by placing $RSRCH$ entirely in OSPF, the address space of OSPF is more unified, which allows better aggregation of its routes. This results in a reduction of the size of $\mathbf{M_X}(EIGRP, OSPF)$ from 9 to 3, which translates to fewer static routes needed, and thus results in less inter-instance complexity (second bar in Fig. 14). In addition, *HD-1* incurs significant intra-EIGRP complexity (first bar), while the actual new design eliminates that complexity.

	static route	redistribution	BGP
default route	38	20	25
redistribution	38	20	25
BGP	43	25	26

Table 8: Complexity associated with different choices of connecting primitive between EIGRP and OSPF. Each cell (*row*, *column*) shows the complexity of the design that uses the *row*(*column*) to send routes from EIGRP (OSPF) to OSPF (EIGRP).

Next, we compare the actual new design and *HD-2*. The main difference between the two is the connecting primitive used to connect OSPF and EIGRP. We found that using route redistribution (i.e.*HD-2*) lowers the complexity compared to using static routes (i.e., the actual new design). This indicates that by changing the connecting primitive from redistribution to static/default routes during the redesign process, the operators introduced unnecessary design complexity.

Given these insights, we next want to find out whether mutual route redistribution is the best connecting primitive to use to connect EIGRP and OSPF, and if alternative primitives could further lower complexity. For this purpose, we enumerate all possible connecting primitives, and apply our framework to estimate the complexity associated with each alternative design choice. The results are shown in Table 8. Note that it is not feasible to use static routes (default routes) to send routes from EIGRP (OSPF) to OSPF (EIGRP), so the corresponding column and row as omitted. The table shows that mutual route redistribution indeed achieves the minimum complexity. A similar complexity could have also been obtained through a design that uses a combination of default routes and route redistribution. We also see that different choices of connecting primitive may lead to significant difference in resulting complexity.

In summary, these results show that (i) the design change of moving subnets of the policy group $RSRCH$ from EIGRP to OSPF greatly reduced both intra- and inter-instance complexity; and (ii) the change of connecting primitive actually made the network more complex and thus should have been avoided; and (iii) different design choices may result in significantly different complexity. Overall, this case study highlights the power of our framework in systematically comparing multiple design alternatives and in guiding operators towards approaches that lower complexity while meeting the same design objectives.

7.3 Operator Interview

We discussed the above results with the operators of the campus network, and they were able to confirm many of our observations. In particular, they confirmed that moving the RSRCH subnets from EIGRP to OSPF significantly reduced the management complexity. In fact, the motivation of that change was to make the RSRCH network more unified and simplify the network design. In addition, the operators also acknowledged that our hypothetical design 2 (Fig. 13) that uses route redistribution instead of static routes could indeed be a less complex design. The primary reason they decided to use static routes in the new design was because this particular operator team consisted of people with varying expertise and skill levels (including senior operators, part-time student workers, and new hires), all of whom could potentially alter configuration files. While configuring static routes did not require extensive prior knowledge, configuring route redistribution required greater knowledge and expertise, particularly given the potential for routing loops. The operators indicated however that they would prefer route redistribution if only a small number of senior operators managed the network. Overall, these results confirm that our framework provides useful guidance to operators. An open question for future work is whether current complexity metrics must be refined to take operator skill levels into account.

8 Discussion and open issues

Incorporating other design objectives and constraints: In putting together a routing design, operators must reconcile a variety of objectives and constraints such as performance, complexity, hardware constraints etc. This paper focuses on the design complexity, given that it is very important, is difficult to quantify, and has received limited attention from the community. In future, it would be interesting to also factor in other important requirements. For example, hardware constraint may restrict the number of route filters that a router can support. Such restriction may in turn impact both intra- and inter-instance route filter placements. We believe our framework can be easily enhanced to systematically determine the best filter placements, so that the hardware constraint is honored, while the total design complexity is minimized. In addition, it may be interesting to consider other design objectives such as performance (e.g., measured as average hop counts between any two subnets), and costs (restricting the number and hardware capacity of devices that can be used). While some of these objectives and constraints may not be critical in a typical over-provisioned enterprise environment, they are nevertheless worthwhile to consider.

Joint optimization of multiple design tasks: This work builds upon a "divide and conquer" network design strategy that is commonly practiced by the operational community [23]. In particular, such a design process consists of four distinct stages: (i) wiring and physical topology design; (ii) VLAN design and IP address allocation; (iii) routing design; and (iv) deployment of services such as VoIP and IPsec. We further break down the task of routing design into two sequential steps: (1) creating routing instances and determining the set of routes to be exchanged between each pair of these instances, and then (2) configuring policy groups and the necessary glue logic. Step (1) is relatively straightforward, typically influenced by factors such as the proximity of routers (e.g., in the same building, city, etc.), administrative boundaries (e.g., different network segments are managed by different operators), and equipment considerations (e.g., EIGRP is available only on Cisco routers). Therefore, this work focuses on the second step while assuming that the first step has been accomplished. In future, it should be beneficial to consider multiple design stages and steps in one framework and explore ways to improve routing design further through joint optimization of all pertinent design choices.

Complexity-aware top-down design: The complexity models presented in this paper pave the way for complexity-aware top-down routing design. Such top-down design takes as input the high-level design objectives and constraints, and seeks to minimize design complexity while meeting other design requirements. In doing so, our complexity models can be used to guide the search of the design space to systematically determine (i) how policy groups should be grouped into routing instances; (ii) optimum placement of route filters; and (iii) what primitives should be used to connect each pair of routing instances. We defer the development of such a top-down design framework to future work.

Emerging architectures and configuration languages: In recent years, researchers have started investigating new network architectures based on logically centralized controllers (e.g., software defined networking [2]), and declarative configuration languages (e.g., Frenetic [11]). These approaches have the potential to simplify network management by shifting complexity away from the configuration of individual devices to programming of the centralized controllers. While these approaches have much potential, hard problems remain such as the need to update network devices in a consistent fashion [22], and building appropriate coordination mechanisms across multiple controllers. Further exploration of the opportunities and challenges of utilizing these new architectures to simplify network design complexity is an important area of future work.

9 Related Work

In recent years, there has been much interest in both industry [1], and academia [5] in developing formal metrics to capture network configuration complexity. We have discussed in detail how our work differs from [5] in Sec. 1. Similarly, our work also differs from other research [7, 15] that measures the configuration complexity in longitudinal configuration data-sets in a bottom-up fashion. There is a considerable amount of prior work on modeling individual routing protocols, particularly BGP [3, 8, 12, 14], and also OSPF [21], to ensure correct, safe, and efficient behaviors from these protocols. There is also recent progress on safe migration of IGP protocols [24] and on modeling the interaction between multiple routing algorithms deployed in the same network [4]. In contrast, our work analyzes how specific routing protocols and primitives should be combined to meet a given set of design objectives, and the focus is on minimizing the complexity of the resulting design. Our notion of policy groups is similar to policy units introduced in [6], but has some differences in that (i) we require subnets within the same policy group to be full reachable to each other; and (ii) we restrict our definition to reachability restrictions on the routing plane since our focus is on routing design, (i.e., we do not consider data-plane mechanisms like packet filters, firewalls, etc.). Algorithms to extract policy units from low-level configuation files were introduced in [6]. In contrast, our focus is on estimating the number of route filters and filter rules, and consequently the resulting configuration complexity, when multiple policy groups are present in a routing instance.

10 Conclusion and Future Work

In this paper, we present a top-down approach to characterizing the complexity of enterprise routing design given only key high-level design parameters, and in the absence of actual configuration files. Our overall modeling approach is to (i) formally abstract the routing specific operational objectives which can help reason about whether and how a combination of design primitives will meet the objectives; and (ii) decompose routing design into its constituent primitives, and quantify the configuration complexity of individual design primitives using bottom-up complexity metrics [5]. We have validated and demonstrated the utility of our approach using longitudinal configuration data of a large-scale campus network. Estimates produced by our model accurately match empirically measured configuration complexity metrics. Discrepancies when present were mainly due to redundant configuration lines introduced by network operators. Our models enable what-if analysis to help evaluate if alternate routing design choices could lower complexity while achieving the same objectives. Analysis of a major routing design change made by the operators indicates that while some of their design changes were useful in lowering complexity, others in fact were counter-productive and increased complexity. Further, our models helped point out alternate designs that could further lower complexity.

Overall, we have taken an important first step towards enabling systematic top-down routing design with minimizing design complexity being an explicit objective. Future work includes modeling a wider range of routing design objectives and primitives (such as selection logic), developing algorithms for automatically producing complexity-optimized routing designs in a top-down fashion, and using similar models to capturing complexity of other enteprise design tasks.

11 Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF) Career Award No. 0953622, NSF Grant CNS-0721574, and Cisco. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or Cisco. We thank Brad Devine for his insights on Purdue's network design. We thank Michael Behringer, Alexander Clemm, Ralph Droms and our shepherd Vyas Sekar for feedback that greatly helped improve the presentation of this paper.

12 References

[1] IRTF Network Complexity Research Group. `http://irtf.org/ncrg`.

[2] Open Networking Foundation. `http://www.opennetworking.org`.

[3] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessensand, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. *Routing Policy Specification Language (RPSL)*. Internet Engineering Task Force, 1999. RFC 2622.

[4] M. A. Alim and T. G. Griffin. On the interaction of multiple routing algorithms. In *Proc. ACM CoNEXT*, 2011.

[5] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. In *Proc. of USENIX NSDI*, 2009.

[6] T. Benson, A. Akella, and D. A. Maltz. Mining policies from enterprise network configuration. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 136–142, 2009.

[7] T. Benson, A. Akella, and A. Shaikh. Demystifying configuration challenges and trade-offs in network-based isp services. In *Proc. of ACM SIGCOMM*, 2011.

[8] H. Boehm, A. Feldmann, O. Maennel, C. Reiser, and R. Volk. Network-wide inter-domain routing policies: Design and realization. Apr. 2005. Draft.

[9] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Take control of the enterprise. In *Proc. ACM SIGCOMM*, 2007.

[10] Cisco Systems Inc. Reliable static routing backup using object tracking. `http://www.cisco.com/en/US/docs/ios/12_3/12_3x/12_3xe/feature/guide/dbackupx.html`.

[11] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: a network programming language. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, pages 279–291, 2011.

[12] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang. Automated provisioning of BGP customers. In *IEEE Network Magazine*, Dec. 2003.

[13] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM Computer Communication Review*, October 2005.

[14] T. G. Griffin and J. L. Sobrinho. Metarouting. In *Proc. ACM SIGCOMM*, 2005.

[15] H. Kim, T. Benson, A. Akella, and N. Feamster. The evolution of network configuration: A tale of two campuses. In *Proc. of ACM IMC*, 2011.

[16] F. Le, G. G. Xie, D. Pei, J. Wang, and H. Zhang.

Shedding light on the glue logic of the Internet routing architecture. In *Proc. ACM SIGCOMM*, 2008.

[17] F. Le, G. G. Xie, and H. Zhang. Understanding route redistribution. In *Proc. International Conference on Network Protocols*, 2007.

[18] F. Le, G. G. Xie, and H. Zhang. Instability free routing: Beyond one protocol instance. In *Proc. ACM CoNEXT*, 2008.

[19] F. Le, G. G. Xie, and H. Zhang. Theory and new primitives for safely connecting routing instances. In *Proc. ACM SIGCOMM*, 2010.

[20] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg. Routing design in operational networks: A look from the inside. In *Proc. ACM SIGCOMM*, 2004.

[21] R. Rastogi, Y. Breitbart, M. Garofalakis, and A. Kumar. Optimal configuration of ospf aggregates. *IEEE/ACM Transaction on Networking*, 2003.

[22] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM*, 2012.

[23] E. Sung, X. Sun, S. Rao, G. G. Xie, and D. Maltz. Towards systematic design of enterprise networks. *IEEE/ACM Trans. Networking*, 19(3):695–708, June 2011.

[24] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure. Seamless network-wide IGP migrations. In *Proc. ACM SIGCOMM*, 2011.