3-24-2010

Formal Analysis of Network Protocols

Anduo Wang
University of Pennsylvania

Formal Analysis of Network Protocols

Abstract
Today's Internet is becoming increasingly complex and fragile. Current performance centric techniques on network analysis and runtime verification have became inadequate in the development of robust networks. To cope with these challenges there is a growing interest in the use of formal analysis techniques to reason about network protocol correctness throughout the network development cycle. This talk surveys recent work on the use of formal analysis techniques to aid in design, implementation, and analysis of network protocols. We first present a general framework that covers a majority of existing formal analysis techniques on both the control and routing planes of networks, and present a classification and taxonomy of techniques according to the proposed framework. Using four representative case studies (Metarouting, rcc, axiomatic formulation, and Alloy based analysis), we discuss various aspects of formal network analysis, including formal specification, formal verification, and system validation. Their strengths and limitations are evaluated and compared in detail.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-10-16.

Formal Analysis of Network Protocols

WPE-II Written Report

Anduo Wang

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
anduo@seas.upenn.edu

March 24, 2010

Abstract

Today's Internet is becoming increasingly complex and fragile. Current performance centric techniques on network analysis and runtime verification have became inadequate in the development of robust networks. To cope with these challenges there is a growing interest in the use of formal analysis techniques to reason about network protocol correctness throughout the network development cycle.

This talk surveys recent work on the use of formal analysis techniques to aid in design, implementation, and analysis of network protocols. We first present a general framework that covers a majority of existing formal analysis techniques on both the control and routing planes of networks, and present a classification and taxonomy of techniques according to the proposed framework. Using four representative case studies (Metarouting, rcc, axiomatic formulation, and Alloy based analysis), we discuss various aspects of formal network analysis, including formal specification, formal verification, and system validation. Their strengths and limitations are evaluated and compared in detail.

Contents

1 Introduction

Today's Internet is increasingly complicated and fragile. Internet forwarding protocols are complicated by middleboxes [27] such as NAT (network address translator), and firewalls which are introduced to address fast growing functionality demands unavailable in the current network hierarchy. As a result, it has became difficult to understand even the elementary concepts such as naming, addressing, and end-to-end connection. On the other hand, in the control plane, the single de-facto Internet protocol BGP (Boarder Gateway Protocol) utilizes routing policies to express the diverse traffic interests of the constituent heterogeneous sub-networks. However, this indispensable policy configuration is error-prone, and misconfiguration at one single network node can cause serious persistent network-wide failures.

Regarding these difficulties, the traditional performance centric bottom-up approach and runtime verification techniques have became inadequate in network protocols development. To cope with the staggering complexity, inherent network heterogeneity, and network scale, there has been growing interests in the use of formal methods to aid in the design, implementation, and verification. With the help of existing formal analysis tools and emerging network models, researchers are beginning to examine network functionality and logical properties to facilitate network protocol development.

In design phase, specialized meta-theory [25, 9, 11] and general logic based formalism [15] are introduced to formally specify fundamental aspects of network functionality such as routing and forwarding. These network-specific formalisms, when combined with external verification tools such as theorem provers [2, 23, 1], model checkers [4], and SMT/SAT solvers [28, 29] enable the formal verification of network standard and design. The resulting sound design can be further used to guide network system development. In addition, lightweight practical validation tools based on logical constraints checking [6], states exploration [22, 16] have been applied to unmodified real-world network implementations to explore network-wide faults.

The rest of the paper is organized as follows. We start with problem statement in Section 2 by reviewing the challenges to today's network. We also present a general formal network analysis framework that accommodates most present-day formal analysis practice throughout network development cycle. Section 3 provides a taxonomy of formal network analysis techniques and existing systems according to the proposed network analysis framework. We then present four representative systems that cover different aspects of formal network analysis: we discuss metarouting in Section 4, routing configuration checker *rcc* in Section 5, axiomatic network formulation in Section 6, and *Alloy* based analysis in Section 7. Finally, Section 8 reviews all the mechanisms and discusses challenges.

2 Problem Statement

To scope our survey paper, we begin with a problem statement by reviewing the challenges and complexity in today's Internet. We then introduce the use of formal networking analysis to aid in network protocol development. We also identify networking properties that are best treated with formal analysis.

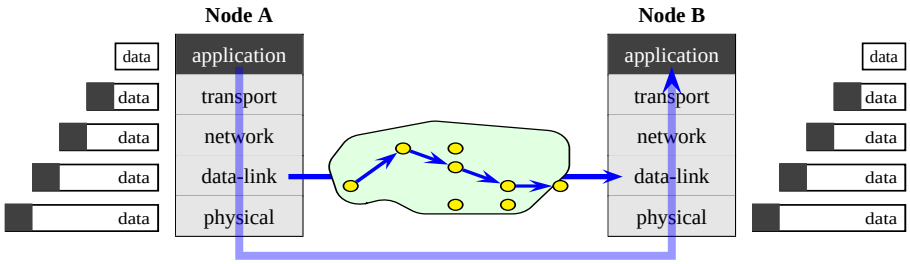

Figure 1: Internet Architecture and Packet Forwarding

2.1 Challenges to Today's Internet

Today's Internet has evolved from the original TCP/IP protocol suite to a global heterogeneous inter-network that provides ever growing network services. The main functionalities involved in any network service include: (1) *addressing* mechanisms that identify network nodes; (2) *routing* protocols that efficient discovers loop-free path discovery through the network; and (3) the actual packet *forwarding* process, implemented at end-nodes (intermediary nodes) with a multi-layer hourglass protocol stack to accommodate network heterogeneity. To review the Internet service architecture, consider a typical packet (message) delivery, shown in figure 1: The message traversal from the creation node A to consumption node B can be broken into the following steps. First, node A needs to identify itself and the destination node by some unique communication addresses; Next, on node A, the message data is passed down the protocol stack where encapsulation at each layer is repeatedly performed by pre-pending new prefix head to carry the corresponding protocol head; Then, based on IP protocol header, the message is forwarded to node B via a number of intermediary forwarding nodes. The actual path along which the message traversed is decided by routing protocols; Finally, node B, upon receiving the message, performs de-encapsulation (removing protocol head at each layer) and passes it up to the application process for interpretation.

Having presented intuitively the concepts of network routing, addressing and forwarding, we now discuss the challenges respectively:

Routing Today's Internet is partitioned into independently administrated autonomous systems (AS). IP routing decides the sequences of intermediary forwarding nodes enroute to the destination node through administrative domains. To express the constituent AS's diverse traffic interests (e.g. competing peers, paid customer-provider relationship, backups service etc.), the interdomain IP routing protocol i.e. BGP (Boarder Gateway Protocol) protocol utilizes rich policy control mechanism based on ranking and filtering (i.e. BGP import and export policy). However, this indispensable routing policy control ability makes BGP staggeringly complicated: BGP policy is essentially configured at and kept private to each AS and the low-level policy configuration process is error prone. The locally distributed policy misconfiguration can cause serious end-to-end connection failures across administrative boundaries; Even worse, though standard IGP (Inter-Gateway Protocols, running within an AS) such as path vector protocol normally comes with convergence guarantee, the static analysis of BGP convergence is NP-hard [10]. As a result, network-wide properties such as network connectivity in today's Internet is hard to understand.

Addressing and Forwarding IPv4 addressing schema associates with each network node a globally unique two-level hierarchical address: The network part of the address identifies the physical sub-network the node belongs to, whereas the host part uniquely identifies the host. This addressing schema has became inadequate with regard to increasing address demands and new functionality. Middleboxes [27] such as NATs are a common solution to alleviate IPv4 address exhaustion. However, middleboxes complicates the current network architecture: The addresses assigned by NAT is no longer globally unique and static. At the same time, middleboxes also complicates protocol layering, the only means for functionality abstraction and modularization in implementing forwarding functionality. In addition, network addressing and forwarding are also obscured by emerging mobile networks, overlay networks etc. Even basic network properties such as end-to-end connectivity are difficult to define and reasoning about.

2.2 Formal Methods in Network Protocol Development

In today's fast growing and increasingly complicated Internet, traditional performance centered bottom-up network engineering is facing unprecedented challenges. The "stimulus-response" style system testing and runtime verification have became inadequate in network development where a single corner case error at one network node can cause serious network-wide failures. With the help of emerging formal network models and established formal analysis tools, researchers are beginning to look at the global logical network properties to facilitate protocol development.

Formal methods are a particular kind of mathematical-based techniques that improve network software qualities with correctness guarantee by rigorous reasoning. Formal methods have been applied throughout the network development cycle. As shown in Figure 2, in design phase, formal specification and verification help deriving sound design , which can be further used to guide the real implementation in formal protocol development. Based on formal specification obtained from network requirement, design, or standard, a full formal verification process can be invoked to establish sound and complete correctness proof. An alternative lightweight analysis-first-prove-last style model-based analysis can be used to find good design instance or counterexamples to facilitate rapid network development. Finally, after system implementation/coding, practical formal validation tools are available to check unmodified system implementation to detect potential system faults against certain properties such as safety, liveness, and other logic invariants.

Before presenting the taxonomy and classification of formal techniques according to the above formal network analysis process, it is informative to first identify some *logical correctness properties* that are particularly suited for formal analysis. Though equally crucial to network correctness, performance related properties and network dynamics are not considered in this survey. In general, the successful delivery of any network service relies on the correctness of network **addressing, routing,** and **forwarding**. Network correctness properties can then be naturally divided as follows: (1) For addressing, the analysis task is to prove that the target addressing schemes continues to provide valid network node address adequate for communication in face of middleboxes and network mobility; (2) For routing, the key problem is to verify that BGP can efficiently discover loop-free routing paths; (3) For the actual packet forwarding within current Internet architecture, the correctness properties address various architectural invariants and forwarding operations.

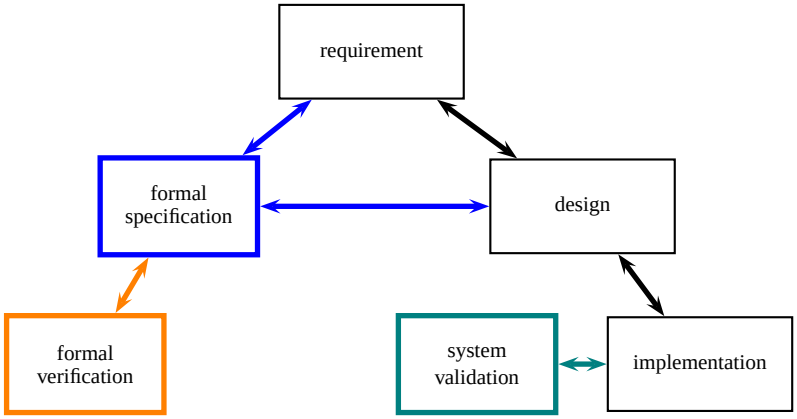

Figure 2: Formal methods in network development

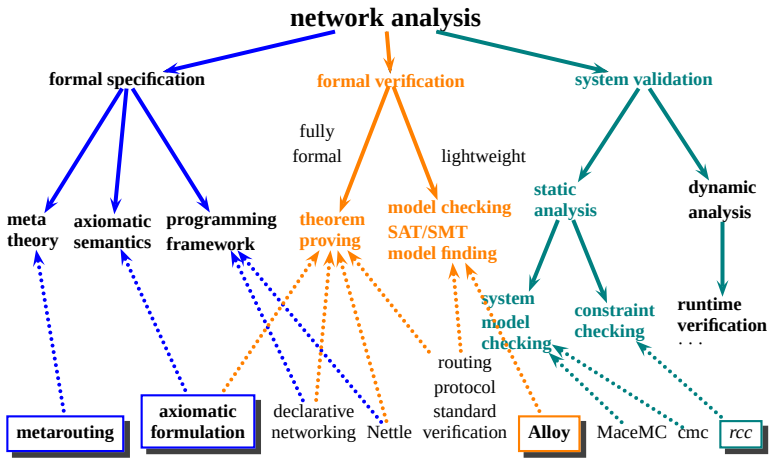

Figure 3: Taxonomy of Formal Network Analysis

3 Taxonomy of Formal Network Analysis

Having presented the major challenges to today's Internet and the proposed formal analysis process throughout network development (Figure 2), we present a classification and taxonomy of formal network analysis. As shown in figure 3, on the top level, based on the application point in network development, formal analysis can be divided into three categories: formal specification, formal verification, and system validation. For each category, we further identify a list of enabling techniques, and representative systems.

3.1 Formal Specification

Taking conceptual network requirement or standard as input, formal network specification can significantly improve the understanding of network requirement/standards,

and help detecting problems early at design phase before investment in implementation. Formal specification often outputs a *formal* design that either meets specified network behaviors and constraints (in a correctness-by-construction) or is of a more checkable/verifiable form for formal verification. Existing formal specification takes one of the following forms: (1) an axiomatic semantics framework with network-specific reasoning support; (2) rigorously defined meta models with added network properties; and (3) programming frameworks that leverage some high-level declarative executable specification language.

The major benefits of formal network specification are as follows: First, the chosen specification formalism not only forces the human user to express the target network problem in a rigorous way (e.g. trivial mistakes can be caught by type-checking), but also supplies/suggests useful conceptual modeling and specification building constructs. Moreover, formal specification, when adapted to a form recognizable by existing verifier/analyzer, enables mechanized correctness checking in formal verification.

Unlike verification and validation, formal specification often operates at a high-level with a focus on pure network functionality and logical correctness, while the low-level implementation complexity (performance, reliability, complication of programming language semantics etc.) is abstracted away. Many formal network analysis efforts are found in formal specification.

Now, we describe representative techniques for the network aspects : routing, forwarding and addressing respectively.

- Routing: The correctness property of Internet (IP) routing as fast convergence and global network connectivity has been intensively studied [25, 5, 3]. Early formal specification of routing utilizes the default logic of existing theorem provers *HOL [3], Nuprl [7], PVS* etc. More recently, network-specific axiomatic semantics frameworks and meta-theories are developed. For example, metarouting [9, 25] etc. introduces the use of algebraic specification model with added convergence guarantee.

- Forwarding and addressing: In contrast to specialized routing model, general-purpose axiomatic framework [15, 13] extended with network forwarding and control primitives are used to specify forwarding and addressing mechanisms. The key challenge is to handle the layered protocol and packet forwarding process implemented at end-nodes as a set of forwarding and control operations. The semantics of the enabling network primitive operations are rigorously given in Hoare-style logic.

Formal Development In addition to formal verification, formal specification often enables formal development by producing a sound guiding design. The problem of preserving the correct design functionality and properties in system implementation is a difficult and open one. Nevertheless, many formal network specification frameworks are integrated with a meta-compiler (interpreter) that helps generate (a large portion of) executable system implementation. An additional key enabling technique is the existence of extensible networking platform (environment) such as Xorp [12] and Click [18], which are originally designed for rapid networking prototyping by providing primitive implementation elements. By utilizes such platforms, the system code synthesize task implemented by the meta-compiler (interpreter) is now reduced to a mapping between formal objects in sound design and the corresponding implementation elements. Finally, moderate inputs on *glue code* are often required to synthesize

the executable system.

The third class of specification technique based on emerging declarative programming frameworks [26, 21, 20, 19] offers an alternative approach by utilizing a high-level logical (or functional) programming language that serves as both the formal specification language and the executable system implementation.

For example, a universal forwarding engine has been synthesized in Click from verified design constructed in axiomatic framework [15]. Similarly, sound design constructed from routing algebra meta model called *metarouting* has also been used to derive routing implementation in Xorp with the help of existing metarouting interpreter.

Obviously, the correctness of the derived system implementation relies on both the design and the translation process implemented by the meta-compiler (meta-interpreter). Though the correctness of meta-compiler (interpreter) is often left un-verified, the formal development process still helps improve the quality of the resulting protocol with added benefits. For example, it enables programmers to focus on high-level system composition in a formal specification language that better relates to desired high-level functionality. Such high-level design decision and structure are often reflected in the derived low-level system implementation.

3.2 Formal Verification

Based on the types of chosen formal specification framework, the following formal verifications are enabled: (1) for logical-based specification framework (e.g. axiomatic semantics) and declarative programming frameworks, the correctness can be established through deductive reasoning in external formal verifier/proof assistant such as theorem prover, model checker or SMT/SAT solvers; (2) for specialized network model, specific correctness properties are automatically derivable in the chosen model. This is also an example of *correctness-by-construction* approach. A major limitation of the above full formal software verification is that it requires high initial investment, expertise in heavy formal methods tools as well as deep understanding of the network problems. Even when the verification succeeds, the verified formal results and arguments are decoupled from real system implementation and can be hardly reused. In general, the extremely expensive non-incremental formal verification is restricted to well-understood network standard and does not scale well to real-world network development.

To mitigate these difficulties, an alternative lightweight approach is emerging in software verification. The idea is to reduce verification efforts to the minimal by developing tools that incorporate specialized moderate meta models and fully-automatic analysis procedures that deal with a subset of verification properties in an analysis-first-prove-last manner. Example tools such as *Z* and *Alloy* [14] have been applied in formal network verification [30] recently.

3.3 System Validation

In contrast to formal specification and verification, static system validation is performed at implementation level, and usually before deployment in real network.

In the first type of static validation based on constraints checking, system faults are explored by checking implementation against correctness properties as invariants or constraints. Validating arbitrary system implementation is hard, formal validation technique often imposes either pre-processing or implementation constraints: (1) to check wider range of unmodified implementation, a pre-processing step is desirable

	Network problems/properties	Techniques/systems
Routing	Specification of routing policies	Nettle [26], **Metarouting** [25, 9], Theorem proving [3, 7], Model checking [3], Declarative networking [21, 20, 19]
	Static BGP convergence analysis	Metarouting, Declarative networking, Theorem proving
	BGP policy configuration faults detection	**rcc**
	Invariants checking of `Chord`	MaceMC [16, 17]
Forwarding	Message deliverability	**Axiomatic formulation** [15]
Addressing	Reachability & returnability	**Alloy** [30]

Table 1: Taxonomy of Networking Problems

to transform the real system into an intermediary and more checkable form; or (2) programming templates or guidelines that constrain the way network system is implemented. The purpose is to help mask irrelevant implementation complexity and extract implementation structures that reflect high-level functionalities of interests. Next, correctness or invariants capturing the correctness properties are evaluated on implementation. In general, the invariants (constraints) are either supplied by network operator or hard-wired in the validation tool, and are usually expressed in some form of logic.

A second static validation approach utilizes exhaustive proof space searching algorithm such as model checking to verify temporal invariants. Note that, though we have included dynamic system validation technique such as runtime verification and testing in Figure 3 for completeness, strictly speaking, they are not considered as formal network analysis.

Routing configuration checker *rcc* [6] is an example of constraints checking based validation tool. *rcc* detects routing policy faults in real-world BGP configurations. *rcc* automatically identifies constraints violation that catches a large set of BGP path and route anomalies. On the other hand, *MaceMC, cmc* [24, 17, 22] are examples of model checking based validation tools. MaceMC, by enforcing the use of its programming templates/guidelines, performs bounded model checking of liveness and safety properties on unmodified network implementations.

Unlike formal specification and validation, to deal with the staggering complexity found only in system implementations, formal validation is usually designed to be fully automatic, and performs on a best-effort basis that is usually neither sound nor complete. The types of checkable properties are also restricted, and real deductive reasoning requiring non-trivial user guidance is not adopted in most cases. Nevertheless, formal validation is an invaluable technique that can be easily introduced to networking developer without inuring extra effort and performance overhead.

3.4 Summary

In Table 1, we highlight the representative network problems and properties studied in present-day formal network analysis for network routing, forwarding and addressing respectively. For each class of network problems and properties, representative

techniques and systems are also listed.

As shown in the table, most formal analysis efforts are found in network routing problems whereas the formal treatment of network forwarding and addressing is only starting to emerge. The table also tells us that for the relatively well-studied routing problem, most formal attempts deal with specification of routing policies, followed by formal analysis of particular properties such as routing convergence, and that only very few practice validation tools are found for specific routing properties. We argue that this is consistent with the development and application of formal analysis in networking: Formal specification which also serves as the basis for formal analysis is the most widely used formal techniques. Based on formal specification that is particularly amenable for analysis, fewer formal analysis tools are developed. Finally, due to the staggering complexity and difficulty in validating real-world routing protocols, even fewer practical validating tools that perform directly on systems implementations are proposed and developed.

It is not feasible to cover all the related literature in this survey. We are going to selectively focus on some representative ones (shown in bold), including: (1) Metarouting, an algebraic framework for routing policy construction with convergence guarantee; (2) *rcc*, a constraints checking based routing policy checker that detects real BGP configuration faults; (3) Axiomatic formulation of networking forwarding functionality; and (4) Addressing analysis in interoperation networks with Alloy.

4 *Metarouting*: Routing Policy Construction with Convergence Guarantee

Metarouting is an algebraic meta-model for routing policy with added property of convergence guarantee. Metarouting attempts to facilitate the design and configuration of routing protocols by providing an algebraic metalanguage that encompasses a large family of routing policies. The major benefit of using metarouting is that the difficult (NP hard [10]) convergence property can be autoamtically derived. Targeted users include: (1) network designer who are interested in the design and development of new routing protocols and modification to existing protocols; (2) network administrators who are responsible for BGP policy configuration to realize the administrative traffic goals and achieve global connectivity.

Metarouting makes the following two contributions:

- Metarouting offers an algebraic metalanguage that captures a large family of routing policies. The metalanguage features two types of algebraic objects: (1) base routing algebras that describe common optimal routing policy and global policy guidelines over single path attribute; and (2) algebra composition operator that generates compound routing policy over multiple path attributes.

- Metarouting metalanguage is shown to preserve routing convergence. First, a specific algebra property called monotonicity is identified and proved to be the sufficient condition for routing protocol convergence. Then monotonicity property is shown to be held by base algebras and preserved by composition operator.

Metarouting algebras with the added convergence property are particularly useful for the specification of BGP policies, the static convergence analysis of which have been proved to be NP-hard. In this survey, we use BGP policies as working examples to illustrate metarouting and its application.

4.1 Background on BGP

Before presenting metarouting specification of BGP policy, it is insightful to briefly review BGP protocol and BGP policy control.

Conceptually, BGP protocol can be decomposed into two routing components: *policy* and *mechanism*. Routing policy describes how routes are measured (i.e. what attributes are attached to each routing path?) and compared (i.e. how route attributes are ranked?). Whereas the routing mechanism (i.e. routing algorithm) maintains adjacent network links, exchanges messages with neighbors, and selects most desirable route upon receiving routing advertisements according to routing policy. In the setting of BGP, routing mechanism part is simply standard path-vector protocol, whereas BGP policy defined over a list of policy attributes is used to express the corresponding AS's traffic goal. Assuming correctness of BGP mechanism, metarouting focuses on specification of BGP policy and convergence analysis.

The common policy control mechanisms BGP offers are: import policies that determine which routes to accept; export policies that decide which routes to re-advertise to neighbors; and the lexicographic routing comparison used in local route selection. In local route selection, for a given destination, each router goes through a list of attributes to compare and select the best route: the router checks one attribute at one time, selects the best route based on that attribute; the router goes down the list and compares the next attribute only if the attributes seen in previous steps are equally good.

4.2 Metarouting Algebras

Metarouting uses abstract routing algebra [25] as the mathematical model for routing policy. An abstract routing algebra A is denoted by a many-sorted algebra tuple

$$A = \langle \Sigma, \preceq, \mathcal{L}, \oplus, \mathcal{O}, \phi \rangle \tag{1}$$

Here Σ describes the set of paths in the network totally ordered by preference relation $\preceq$. Intuitively, the preference relation $\preceq$ is used by the routing protocol algorithm to compare and select the most desirable route (path); $\mathcal{L}$ is a set of $labels$ describing links between immediate neighbors. Note that the labels may denote complicated policies associated with the corresponding link; $\oplus$ is a mapping from $\mathcal{L} \times \Sigma$ to Σ, which is the *label application operation* that generates new paths by concatenating existing paths and adjacent links; $\mathcal{O}$ is a subset of Σ called *origination* that represents the initial routes stored in the network; Finally ϕ is a special element in Σ denoting prohibited path that will not be propagated in the protocol.

Base Algebras Based on the notion of abstract routing algebra, metarouting offers two metalanguage features to construct algebra instances (interpretations). The first language feature is called base algebras that can be directly instantiated to model single-attribute policies for various optimal path policies and guidelines.

As a first example, shortest path policy defined over routing path cost (path attribute interpreted by an integer cost) selects routes with lower cost (preference relation interpreted as the normal ordering $\leq$ over integers). Formally, shortest path policy can be represented by the following algebra instance:

Σ: describes routing paths	path cost
$\preceq$: preference relation over Σ	$\leq$
$\mathcal{L}$: link label	link cost
$\oplus$: function from $\mathcal{L} \times \Sigma$ to Σ	$+$
$\mathcal{O}$: origination routes	path cost in initial routing table
ϕ: prohibited path	16 (discard longer paths)

Here, the prohibited path is instantiated to integer 16, which implies that no path with costs more than 16 is considered in the protocol[1]. In metarouting, this algebra instance can be obtained directly from a pre-defined base algebra `ADD(n,m)` by instantiating n to 1, and m to 16.

We now look at a second example use of metarouting base algebras. Gao-Rexford policy guideline is well-known for its convergence guarantee. It also reflects an ISP's incentive to reduce the use of provider routes and encourage the use of customer routes for economical reasons. Like shortest path, Gao-Rexford policy guideline can be expressed by an algebra instance as follows:

Σ: routing paths	$\{C, R, P, \Phi\}$
$\preceq$: preference relation	$C \preceq R, \quad R \preceq P, \quad C \preceq P$
$\mathcal{L}$: link label	$\{c, r, p\}$
$\oplus$: $\mathcal{L} \times \Sigma \mapsto \Sigma$	$c/p/r \oplus * = C/R/P$
$\mathcal{O}$: origination routes	route path type $(C/R/P)$ in initial routing table
ϕ: prohibited path	Φ

Here routing paths (links) attributes are interpreted as an enumerated type $\{C, R, P\}$ ($\{c, r, p\}$) denoting a customer, peer, or provider path (link). And preference relation is interpreted to reflect the ISP's preference favoring customer and peer routes over provider routes. Like shortest path, this algebra instance can be obtained by instantiating metarouting base algebra `lp(3)` [2].

Lexical Product Composition So far we have seen the use of metarouting base algebras to construct simple policies defined over single route attribute. We now describe the construction of compound BGP policies over multiple route attribute by using the second metarouting metalanguage feature: composition operator called lexical product $\otimes$.

As its name suggests, lexical product operator models the lexicographical comparison used in BGP route selection described in 4.1. The intuition is that a lexical product algebra $A_{lex} = A_1 \otimes A_2$ constructed from two sub-algebras A_1, A_2 models a BGP policy with two group of attributes, where the more important attributes are handled by the first sub-algebra A_1, and those less important used to break tie in route selection are handled by the second sub-algebra A_2. Formally, the preference relation $\preceq_{A_{lex}}$ of A_{lex} is defined by the constituent sub-algebras preferences $\preceq_{A_1}$ and $\preceq_{A_2}$ as follows:

$$\langle \sigma_1, \sigma_2 \rangle \preceq_{A_{lex}} \langle \beta_1, \beta_2 \rangle \equiv \sigma_1 \preceq_{A_1} \beta_1 \vee (\sigma_1 \sim_{A_1} \beta_1 \wedge \sigma_2 \preceq_{A_2} \beta_2) \qquad (2)$$

As an example use of lexical product composition, consider a BGP policy that is compliant with Gao-Rexford policy guideline described in the previous section. A

[1] This bound 16 is used as a prevention of count-to-infinity problem

[2] An additional rewriting $C/c \to 1, R/r \to 2, P/p \to 3$ is required

possible metarouting algebra representation can be written as $A_{compliant} = A_{guide} \otimes A_{other}$ where $A_{guide} = lp(3)$ denotes the algebra instance for Gao-Rexford guideline, and A_{other} models the remaining aspects of the BGP policy. Obviously $A_{compliant}$ models a guideline compliant policy because the first sub-algebra A_{guide} ensures Gao-Rexford guideline is always enforced.

4.3 Metarouting Convergence Property

By adopting the use of routing algebra as the mathematical model for routing policy, metarouting reduces policy properties/constraints to algebra proprieties. For example, the behavior of prohibited path as the least preferred path that is closed under path concatenation is captured by algebra properties [25] *maximality* and *absorption*.

Moreover, the difficult convergence analysis is reduced to the examination of algebra *monotonicity* property. Algebra property *monotonicity* imposes the restriction that a route becomes less preferred when it "grows" (i.e. when route concatenation occurs), as illustrated in its definition:

$$Monotonicity: \qquad \forall_{l \in \mathcal{L}} \forall_{\alpha \in \Sigma} \quad \alpha \preceq l \oplus \alpha \tag{3}$$

It is proved in [25] that monotonicity is a sufficient condition for BGP system convergence:

Theorem 1 *A BGP system is guaranteed to converge if the BGP policy can be modeled by a strict-monotonic algebra*

By this theorem, we can conclude that both shortest path policy and Gao-Rexford guideline ensure convergence because they can be modeled by monotonic metarouting algebras. This is consistent with the convergence results obtained from manual proofs where all possible BGP executions are enumerated.

To help user predicate system convergence, the monotonicity of all metarouting base algebras are given. In addition, it is shown that lexical product composition preserves monotonicity. That is, as long as the first sub-algebra is monotonic, the resulting lexical product algebra is monotonic regardless of the rest sub-algebras. Recall the Gao-Rexford guideline compliant BGP policy modeled by $A_{compliant} = A_{guide} \otimes A_{other}$. Because lexical product preserves monotonicity, and that we know $A_{compliant}$ is also monotonic regardless of the monotonicity properties of A_{other}. By theorem 1, we can conclude that any Gao-Rexford guide compliant policy $A_{compliant}$ ensures BGP convergence regardless of the rest of the policy modeled by A_{other}. This is also consistent with the major convergence result in [8].

4.4 Evaluation

According to the proposed taxonomy shown in Figure 3, metarouting is an example of formal specification technique with added routing protocol convergence property. As shown in Figure 2, metarouting takes conceptual requirement as input, and allows the user to construct "convergence guaranteed" routing protocol policies in the form of metarouting algebras. Metarouting can be potentially used to derive sound BGP system implementations with the help of metarouting meta-interpreter. By autoamtically deriving "convergence guaranteed", additional verification effort is no longer required. As a result, metarouting can also be viewed as an example of correctness-by-construction routing design. We summarize strengths and limitations of metarouting as follows:

Strengths

- Convergence by Construction: Unlike previous efforts of combinatorial model and analysis, metarouting is the first BGP system model with convergence guarantee.
- High-level modular policy configuration language: Metarouting algebras serve as a policy configuration language that enables the human user to focus on the high-level routing policy composition without worrying about the low-level policy configurations and their convergence properties. The pre-defined metarouting base algebras and composition operators can also be viewed as a default modular library that eases algebraic policy construction and encourages code and formal argument reuse.

Limitations

- Limited expressive power: Metarouting relies on *monotonicity*, a sufficient but not necessary condition to provide convergence guarantee. Therefore, metarouting is not complete with regard to convergence. And metarouting does not address well-converging routing protocols built from certain non-monotonic attributes such as `MED`, even though they provide useful semantics in practice.
- Ranking and filtering: Though metarouting is a natural fit for various optimal path vector protocol and BGP policy guideline, it is not clear how metarouting can be used to model important BGP import/export policies based on router-specific ranking and filtering. The difficulty arises from the fact that metarouting algebra is per-AS based, as we will elaborate in the next paragraph.
- Global algebra vs local policy configuration: By default, routing algebra is by natural global. That is, all routes traversing the network (within an AS) are measured with regard to one single global algebra. The various optimal path policies are such example global algebras. Another example algebra is for policy guideline which, though allows some flexibility, nevertheless, assumes global coordination/agreement among all distributed routers. This causes difficulties in Internet routing (inter-domain routing protocols such as BGP in particular) practice: (1) For competing reasons etc., Internet routing policy is configured and kept private locally at each AS; (2) Even within one single domain where an network operator has global access, the operator may want to configure routers in this same domain differently (e.g. different ranking/filtering policies at each router).

5 Detecting BGP configurations faults with *rcc*

rcc is a router configuration checker for real BGP systems. *rcc* detects BGP routing faults that can potentially cause persistent routing failures by checking BGP configurations against a set of pre-defined high-level correctness constraints. *rcc* is designed for before deployment. *rcc* is intended to be used by network operator within one single AS.

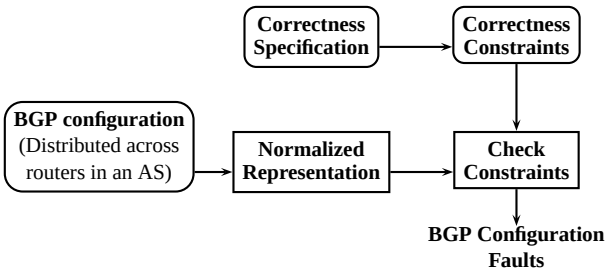

Figure 4: Overview of rcc [6]

5.1 Overview

rcc checks real BGP configuration distributed within a single AS. As shown in Figure 4, *rcc* functionality can be decomposed into two parts: (1) a pre-processor and parser that converts input vendor-specific BGP configuration into an intermediary normalized representation, (2) a constraints checker that performs the actual checking on the normalized configuration according to a set of pre-defined high-level correctness constraints derived from a correctness specification. The goal of *rcc* is to provide *before-deployment* correctness checking ability and helps network operators move away from the "stimulus-response" reasoning.

5.1.1 Configuration Preprocessing and Parsing

Before constraints checking, *rcc* pre-processes the vendor specific BGP configurations distributed within the AS, and produces an intermediary more-checkable normalized representation. Basically, by de-referencing policy references such as filters, *rcc* parser builds the normalized representations as a set of relational database tables. Note that, *rcc* keeps track of all normalized policies: the policies with same operations, addressed by different names, or even when implemented in different configuration languages, are recognized as the same policy.

5.1.2 Correctness Specification and Violation

rcc detects BGP faults that cause persistent network-wide failures by checking BGP configuration against correctness specification. Built-in *rcc* correctness specification identifies two types of high-level correctness properties: *path visibility* and *route validity*.

Path visibility property asserts that for any reachable destination, at least one usable path will be discovered. A usable path is one that en-route to the correct destination node which is also compliant with BGP policies along the path. Specifically, in the scope of a single AS, path visibility ensures that the BGP configuration enables the propagation of routes to all reachable external destinations with regard to the intended BGP policy. Example path visibility violation is caused by iBGP misconfiguration which, when combined and interacted with IGP, prevents the dissemination of routes to external destinations.

Route validity property asserts that any computed route corresponds to an actual usable path along which packets can be successfully forwarded to the destination.

Intuitively, these two properties ensure that a router is always capable of finding a usable route to a destination when there exists one path conforming to the intended policy; and that each route the router discovers always reflects an usable underlying physical path. However, for verification purpose, these properties do not automatically suggest what conditions (constraints) shall be checked in real configuration. And the derived constraints enabling the actual faults detection is addressed in the next section.

5.1.3 Correctness Constraints and Faults Detection

For each correctness property, *rcc* identifies a set of constraints (conditions) that the BGP configuration shall conform to. These constraints (conditions) are then evaluated on the normalized policy database representation (produced by *rcc* preprocessor) to detect potential misconfiguration.

Path visibility ensures BGP policy does not create network partitions in a connected network (in lower layer). In particular, for iBGP configuration, it implies that within the AS, each internal router will eventually learns the routes to all reachable external destinations through the eBGP router. The *iBGP signaling partition* problems caused by iBGP misconfiguration when interacted with IGP protocols is the focus of *rcc* fault detection with regard to path visibility. Specifically, *rcc* checks iBGP configuration with and without route reflector (RR) as follows:

- In the absence of RR, a (sufficient) trivial constraint requires that the iBGP topology be a "full mesh".
- In practice, a full mesh topology is seldom adopted, instead, RR is used to improve scalability. In the presence of RR, the constraint requires only a full mesh topology among non-RR client BGP routers.

The actual *rcc* constraints checking of the above conditions is performed by iBGP signaling graph construction from the normalized policy tables.

Route validity The challenge in route validity checking is the detection of *policy-related* violations. Without requiring additional human user's input on the intended policy, *rcc* checks against the proposed "policy belief" in compliance of best common practice. Example constraints imposed by best common policy belief include:

- Routes learned from a peer are not re-advertised back.
- Configuration anomalies are likely to be mistaken: for a given destination, when configurations at different routers differ, the few deviations are likely to be mis-configured.

5.2 Evaluation

According to the proposed taxonomy shown in Figure 3, *rcc* is a static analysis tool detecting routing protocol configurations faults. As shown in Figure 2, *rcc* checks the unmodified BGP configurations against built-in correctness constraints. We summarize strengths and limitations of *rcc* as follows:

Strengths

- Lightweight static analysis of real-world BGP configuration: *rcc* reduces the analysis effort/burden imposed on network operator to the minimal. Unlike most formal analysis tools, *rcc* does not require any expertise/experience in formal methods tools/techniques, nor any human interaction/inputs (not even the specification of the network operator's intended policy!) in the reasoning/checking process; Yet like all static analysis tool applied before deployment, *rcc* does not incur any performance overhead, and can detect problems in unmodified real-world vendor-specific BGP configurations.
- Checking network-wide properties at local AS: *rcc* enables network-wide faults detection of routing configuration by performing constraints checking at the local AS.

Limitations

- Soundness and completeness: Like many practical static analysis tool, *rcc* is neither sound nor complete. The proposed constraints derived from path visibility and router validity properties catch only a sub-set of potential active faults which, when triggered by certain event sequences, will cause end-to-end persistent route failures. On the hand, constraints checking may also report false positive violations, which do not correspond to any route failures.
- Best Common policy checking: To reduce user interaction to the minimal, *rcc* operates without the intended policy specification, and relies on proposed "policy beliefs". Such coarse-grained treatment of "intended policy" introduces false-positives constraint violations as well as the mask of potential faults in the real intended policy that is abstracted away.

6 Axiomatic Basis for Communication

A first axiomatic semantics framework [13] for network service is proposed in [15] within the current layered Internet architecture. The intended usage of this semantics framework includes: (1) specification framework for the understanding, specification and reasoning of the increasingly complex Internet architecture and forwarding functionalities; (2) sound design language that facilitates rapid implementation of network protocols.

The axiomatic semantic framework first provides a metalanguage for the specification of common network functionality such as addressing, naming, forwarding schemes, and the expression of architectural invariants such as message delivery. The semantics of this metalanguage is then rigorously defined in Hoare-style logic extended by "leads-to" relation which denotes the central network functionalities: "store-and-forward". This Hoare-style semantics also serves as the foundation for network verification.

6.1 Basics: Abstract Switching and Forwarding

A central problem in today's Internet is the design and implementation of packet switcher and the associated functionality: package forwarding. The modeling and reasoning of

most Internet services at different network layers rely on the notion of forwarding. Therefore it is desirable to have a proper abstraction of the switching unit and forwarding functionality. In this axiomatic framework, the notion of the "store-and-forward" switcher is captured by an abstract object called *Abstract Stitching Element* (*ASE*), and the forwarding principle is modeled by "leads-to" relation.

An abstract switching element (ASE) generalizes the notion of the switcher, which may be called with different names at different network layers such as switcher, bridge, gateway, or router etc. Each ASE A is associated with two types of logical ports: input ports ${}^{x}A$ that receives packets from a predecessor ASE x; and output ports A^{x} that forwards message to the successor ASE x. Here variable x is used to range over predecessor ASEs and successor ASEs. For example the logical ports for ASE B placed between A and C are written as: ${}^{A}B, B^{C}$. Specifically, ${}^{0}A$ and A^{0} denote the two end ports where a message is created and consumed respectively (recall nodes A and B in Figure 1). The communication message m at a port x is denoted by $m@x$. And the switching table S_B maintained at ASE B is modeled by a set of mappings: $\langle A, p\rangle \mapsto \{\langle C, p'\rangle\}$ between $\langle ASE, header\rangle$ pairs. Switching table lookup at ASE B for message received from A with header p is written as $S_B[A, p]$.

On the other hand, "leads-to" relation $m@x \rightarrow n@y$ models the central forwarding operation which can be read as a message m at port x is forwarded to port y as message n. The communication between directly connected ASEs A, B can be simply written as ($m@A^{B} \rightarrow m@^{A}B$). Obviously, to accommodate message forwarding through intermediary nodes, "leads-to" must satisfy transitivity property:

$$\forall x, y, z, m, m', m'' : (m@x \rightarrow m'@y) \wedge (m'@y \rightarrow m''@z) \implies m@x \rightarrow m''@z$$

The actual message switching operation at each intermediary node B (from predecessor ASE A to successor ASE C) can be formalized as follows:

$$\forall A, B, C, m, p, p' : \exists {}^{A}B, B^{C} \wedge \langle C, p'\rangle \in S_B[A, p] \implies pm@^{A}B \rightarrow p'm@B^{C}$$

Here $S_B[A, p]$ denotes the switching table lookup operation at ASE B for message received from predecessor ASE A with prefix p. After the switching table lookup, B transforms the prefix p to p' as specified by the chosen forwarding protocol and forwards the resulting message $p'm$ to successor ASE C according to $S_B[A, p]$.

By adopting the above abstract notions of ASE and "leads-to", common communication concepts can be concisely formalized. For example, communication *address* used to identify a network entity can be specified as follows:

Address If $\exists$ ASEs A, B and prefix $p \neq \emptyset$ such that
$\forall m : \ pm@^{x}A \rightarrow pm@^{y}B \rightarrow m@B^{z}$ then p is an address for B at A

Here the intuition is that the address which identifies the destination node B is the prefix that will not be changed along the forwarding path towards B, and that after arriving at B, the address p will be removed from the carrying message m for further processing.

6.2 Meta-language for Packet Forwarding

We have shown that based on the notion of ASE and "leads-to", common communication concepts can be easily formulated. To further facilitate the development and reasoning of real forwarding practice, the axiomatic framework identifies a set of primitive

operations to serve as a meta specification language. Besides forwarding, axiomatic framework also includes primitives for control mechanism, which determines the actual path along which message is forwarded, as described in section 2.

Forwarding Primitives To forward message (i.e. realization of the "leads-to" relation) among two end-node in today's Internet, as shown in Figure 1 (end-nodes A and B), repeated message *encapsulation* at message creation node A and *de-encapsulation* at message consumption node B are performed. At each node (modeled by ASE) along the forwarding path, switching lookup and the actual *forwarding* are performed. Besides packet forwarding, to accommodate virtual circuit networks and recent middleboxes extensions such as NAT, firewall, additional operations are also included. The following primitives capture these necessary functionality:

- Encapsulation and de-encapsulation are performed by primitives `push(message, string)`, and `string pop(message)`.
- The actual store-and-forward is performed by primitives `send(ase,message)`, `<ase,message> receive()`, `lookup(ase,string)`, and `message copy(message)`.

Control Primitives Routing is the control process that decides along which nodes and paths messages are forwarded. Specifically, switching (forwarding) table is maintained by routing protocols. The routing challenges addressed in the axiomatic framework include: (1) naming (address carried by protocol head) in overlay network and inter-networks connected by gateway; (2) specific path setup mechanism for virtual-circuit (swap) and Ethernet (bridging). Accordingly, axiomatic framework supplies the control primitives to implement these functionalities. Intuitively, the control primitives augment the ASE formalism with switching table update ability and control message exchange ability operations

- Primitives for switching table updates are: `update(ase,string,ase,string)`
- Primitives for control message path setup include: `string getlabel(message)`, `setlabel(message, string)`, `message create(opcode)`, and `message response(message, opcode)`

Combining the above forwarding/control primitives and the usual control flow primitives in procedure languages (condition, branch, loop), typical processing patterns for forwarding data, resolution, path setup etc. can be concisely specified. For example, the regular data message forwarding can be implemented by primitives as follows:

```
1        string n = pop(msg);
2        {<ase, string>} S = lookup(prev, n);
3        for each <ase, string> s_i in S {
4                message outmsg = copy (msg);
5                push (outmsg, s_i.string);
6                send(s_i.ase, outmsg);
7        }
```

Intuitively, the above program specifies that upon receiving a message `msg` at an intermediary ASE, forwarding primitive `pop` is first invoked to get the outermost name

(destination address), followed by a switching table `lookup` operation to decide the outgoing successor ASEs (denoted by `S`). Next, for each successor ASE `s_i`, the message data is copied and stored at `outmsg`, and new prefix is generated by `push` primitive in line 5. Finally, `send` primitive moves the message to the corresponding successor ASEs.

6.3 Formal Semantics in Hoare Logic

To facilitate formal reasoning, a rigorous formal semantics of the proposed primitives are given in Hoare-style logic. Basically, the semantics of each primitive operation is given by a pair of pre/post-condition assertions. And the pre/post-assertion are specified in the usual first-order logic augmented with "leads-to" relation.

For example, the meaning of primitive `send` is captured by the following axiom:

$$\begin{array}{rcl} \texttt{send(C,m)} & & \overline{\varphi_1\{\texttt{send(C,m)}\}\varphi} \\ \text{where } \varphi & = & \theta \supset m'@x \rightarrow m''@y \text{ and} \\ \varphi_1 & = & \theta \supset (m'@x \rightarrow m''@y \vee (m'@x \rightarrow m@A^C \wedge m@A^C \rightarrow m''@y)) \end{array}$$

Here for the post-condition φ to be true, which asserts that message m' can be forwarded from port x to y (i.e. "leads-to" relation $m'@x \rightarrow m''@y$), the pre-condition is that either the "leads-to" relation is *already* true regardless of `send` or that the "leads-to" relation is established by `send(C,m)` operation. In the later case, the intuition is that `send(C,m)` is performed by an intermediary port (A^C) in multi-hop message forwarding. As a second example, the meaning of `push` is captured by Hoare-style axiom extended with "leads-to" relation as follows:

$$\texttt{push(m,n)} \qquad \overline{\varphi[nm/m]\{\texttt{push(m,n)}\}\varphi}$$

Here, `push` operation that models prefix encapsulation is simply interpreted as prefix prepending (prepend `m`, the message being carried, with a new prefix `n`). The formal semantics of other primitives are given in a similarly way.

An Example Proof Equipped with the formal semantics and the underlying extended Hoare-style logic, formal verification is enabled: to perform verification for a given property of a service specified in the axiomatic framework, simply formulate the desired property by a proper post-condition, and repeatedly apply predicate-transformation for each primitive being executed according to Hoare logic in the reverse order the primitives are executed. Correctness is established by reducing the post-condition to a trivially-true pre-condition.

As an example proof, consider the 7-line message-forwarding program in the previous section. Without loss of generality, assume the program is executed at ASE A, and the message `msg`$= pm_0$ (p is the prefix, and m_0 is the message data being carried) is received from port ${}^BA$ connecting to some predecessor ASE B. First, the user comes up with the property of interests, i.e. the desired post-condition as follows:

$$\begin{array}{c} \text{For each entry for } \langle B, p\rangle \text{ in } A's \text{ switching table :} \\ \bigwedge_{s_i \in S} \supset (pm_0@{}^BA \rightarrow (s_i.string)m_0@A^{s_i.ase}) \end{array}$$

This post-condition says the intended effect of the forwarding program is that received message `msg`$= pm_0$ can be successfully forwarded to all successive ASEs $s_i.ase$ according to the switching table entry S for the received message $\langle B, p\rangle$. To verify this post-condition, one tries the Hoare-logic axiom for each executed operation in a backward fashion. First, as shown in the last line, } denotes the outermost `for` loop for each successive ASE in the switching table. By applying the Hoare-Logic axiom for `for` loop, the post-condition is reduced to the following form:

$$s_i \in S \supset (pm_0@^B A \rightarrow (s_i.string)m_0@A^{s_i.ase})$$

Next, primitive `send` is executed (line 6), by applying the corresponding axiom `send(C,m)` introduced in the previous section, we can reduce the post-condition to the following form:

$$\begin{aligned} s_i \in S \quad \supset \quad & ((pm_0@^B A \rightarrow outmsg@A^{s_i.ase} \wedge outmsg@A^{s_i.ase} \rightarrow \\ & (s_i.string)m_0@A^{s_i.ase}) \vee \cdots) \end{aligned}$$

Note that we only consider the second branch in `send(C,m)` axiom where the execution of `send` is critical to reduce pre-condition. The first irrelevant branch is therefore denoted by $\cdots$. This proof choice is non-trivial and indeed requires user direction. In more complicated cases, genuine insights in the proof process are often needed, and though the formal semantics introduced in the axiomatic framework make the proof mechanically checkable, the axioms themselves do not suggest how a proof can be constructed. Following this backward post-condition to pre-condition transformation, we see primitive `push` executed in line 5, accordingly apply axiom `push(m,n)`, and the new pre-condition is derived:

$$\begin{aligned} s_i \in S \quad \supset \quad & (pm_0@^B A \rightarrow (s_i.string)outmsg@A^{s_i.ase} \wedge \\ & (s_i.string)outmsg@A^{s_i.ase} \rightarrow (s_i.string)m_0@A^{s_i.ase}) \end{aligned}$$

Similarly, repeat such predicate transformations according to the Hoare-logic axiom associated with primitives in line `4-1` as well, one finally arrives at the following pre-condition:

$$\begin{aligned} s \in S_A[prev, p] \qquad \supset & (pm_0@^B A \rightarrow (s.string)m_0@A^{s.ase} \wedge \\ & (s.string)m_0@A^{s.ase} \rightarrow (s.string)m_0@A^{s.ase}) \end{aligned}$$

This pre-condition is trivially true from assumption:

$$prev = B \wedge m = pm_0 \wedge \exists^B A$$

Which simply says B is the predecessor ASE and `msg` is of the form pm_o.

6.4 Evaluation

According to the proposed taxonomy shown in Figure 3, the axiomatic framework discussed in this section is an example of formal specification technique for network forwarding. Similar to metarouting, this semantics framework facilitates the understanding, reasoning, and design of network protocols, as shown in 2. The major difference is that metarouting enables "specific soundness property" (i.e. convergence) by correctness-by-construction, whereas axiomatic formulation, by defining an rigorous semantics in Hoare-style logic, establishes "a wider range of soundness properties" by formal reasoning. We summarize the strengths and limitation of axiomatic formulation as follows:

Strengths

- A general purpose high-level logical specification language: The proposed meta-language is rich and flexible enough for the expression of network service, protocol properties, and architectural invariants. In addition, emerging features, though violating the original Internet design principles, such as middleboxes, can be easily specified. With the help of a meta-compiler, a preliminary universal forwarding engine has been implemented in Click.
- Verification foundation: The formal semantics in Hoare logic with "leads-to" relation serves as a rigorous foundation for formal network verification.

Limitation Internet today is a huge and complex system. The axiomatic framework is among the initial efforts towards an integrated environment that provides a precise model, modular formal development and reasoning support. To outperform or compete with existing network programming framework with pre-implemented network primitives and various automated formal reasoning tools, a real compiler that helps synthesize system codes and automated reasoning procedures are desirable.

7 Addressing Analysis with *Alloy*

Alloy is a lightweight integrated tool for object-oriented style formal specification and automatic analysis. In [30], Alloy is used for addressing analysis in incorporating networks. The major benefit with Alloy is that: the Alloy specification language eases the formalization of addressing and interoperation requirement and Alloy analyzer takes care of the analysis process.

7.1 Overview

Network addressing and interoperation are two enabling mechanisms for global network connectivity. Alloy analysis tool featuring a friendly relation logic language and model-finding based analyzer is used to better understand the addressing problem in interoperation networks.

7.1.1 Connection and Interoperation Specification in Alloy

As a first step, an abstract model for network notions such as connection and interoperation are constructed in Alloy specification logic. These formal specification will be used as the basis in Alloy analysis described in the next section. Alloy features a specification language that combines relational logic and fragment of second order logic. This specification language enables abstract modeling in an object-oriented style augmented with flexible use of logical quantifiers. We summarize the Alloy abstraction (specification) for connection and interoperation as follows:

Connection

- Agents: Alloy specification for agent models network end nodes that are either a *client* or a *server*.

- Domain: Alloy specification for domain models the addressing mechanism that assigns for each participant agent an *address*. Note that an agent can participant in multiple domains and therefore obtain different addresses for different domains.
- Hop: Alloy specification for hop models a particular persistent *connection* between two agents through a common domain both of the agents participate.
- Link: connects adjacent hops to create multi-hop connection.

As an illustrating example, the Alloy specification for `Domain` is as follows:

```
sig Domain {space: set Address, map: space -> Agent}
fact{all d: Domain, g: Agent |
        g in Address.(d.map) => d in g.attachments}
```

Here, the `signature` statement says each `Domain` contains a set of addresses and maintains a mapping between the addresses to agents. Recall that in Alloy, an agents is either a client or server attached to some domain. Next, the `fact` statement asserts the constraint as part of `Domain` specification that an agent is attached to a domain as long as its address is maintained by that domain.

Interoperation

- Feature: models abstract service which is deployed in a domain and implemented by some *servers* (i.e. agents in the domain that provide the abstract service).
- Interoperation: is a special service, and the server that implements interoperation service models *gateway*.

As an illustrating example, the Alloy specification for `Feature` is as follows:

```
abstract sig Feature {domain: Domain, servers: set Server}
        {some servers}
fact {servers: Feature one -> Server}
```

The intuition in `sig` statement is that a `feature` is always deployed in a particular domain, and provided/implemented by a non-empty set of servers. The following `fact` statement asserts that server can be viewed as a function from `features` to `servers`. Interestingly, based on this Alloy specification, a dynamic, single-address NAT can be viewed as an interoperation service that maps many private IP addresses to a single public one.

7.1.2 Requirement Analysis in Alloy

Based on the abstract model above, Alloy analyzer is used to facilitate user-level interoperation requirement analysis. The two requirement properties considered are *reachability requirement* and *returnability requirement*

- Reachability: reachability property asserts that if a domain assigns an addresses to a client for reachability service, then the client can be reached by that address.
- Returnability: returnability property of connection says that if a client A can be reached by some source client B, then client A should also be able to request connection to reach client B that initiates the first connection.

To analyze these two properties in Alloy, they are first encoded as additional Alloy constraints. As an illustrating example, the Alloy encoding of `reachability` is as follows:

```
assert Reachability {all c: Connections,
        g1, g2: Client, h: Hop, a: Address, d: Domain |
                g1 = h.initiator && d = h.domain &&
                a = h.target && (a->) in g2.knownAt
        => (some h2: Hop | g2 = h2.acceptor && (h->h2)
                in c.connected)}
```

The intuition here is that if a client `g1` requests a connection to a reachable second client `g2` by address `a` that is in `g2`'s `knownAt` set, then `g1` is connected to `g2` through some hop `h2`.

Based on the above Alloy constraints, Alloy analyzer then searches for network instances that meet the requirement constraints, as well as counter-examples. For a moderate sized model consistent with requirement constraints, Alloy returns a model (satisfying instance) that is very convincing and convenient. In general, Alloy performs constraints checking by exhaustive model finding up to a fixed bound.

7.2 Evaluation

According to the proposed taxonomy shown in Figure 3, Alloy based addressing analysis is an example of formal verification of network requirement. As shown in Figure 2, Alloy takes conceptual requirement as input, and requires user effort to construct a formal model for addressing and interoperation in Alloy specification language. Next, Alloy takes care of constraints analysis based on the formal specification. We summarize strength and limitation of Alloy as follows:

Strength Alloy specification language and integrated analyzer significantly help the understanding of the confusing addressing problems in interoperation networks. In contrast to a full formal verification, the automatic Alloy analyzer relying on model finding techniques rather than deductive reasoning has been proven to be sufficient for reasoning about addressing.

Limitation Like many formal verification tools, Alloy requires an initial investment in formal specification before automatic analysis can be performed. The model-finding (through model enumeration process) analysis, though sufficient for the specific case study of addressing, when compared with traditional deductive reasoning, may become less convincing for more complicated analysis tasks. Finally, the analysis is restricted to an abstract model built-in Alloy specification language that is decoupled from real world implementation.

8 Discussion

In this section, we conclude our survey with comparisons of the different techniques proposed for formal network analysis, followed by a discussion of the challenges.

	Metarouting	**Axiom formulation**	**rcc**	**Alloy**
Application scope	Specification	Specification & verification	validation	Specification & verification
Expressiveness	Medium	High	Low	Medium
Automated reasoning support	High	Low	High	High
Soundness	Yes	Yes	No	Yes
Completeness	No	Yes	No	No
Closeness to implementation	Low	Medium	High	Medium
Initial investment in system modeling	Medium	High	Low	High

Table 2: A Summary of the evaluations for different network analysis systems

8.1 Comparison

In previous sections, we have discussed representative techniques and systems for formal network analysis, including metarouting framework for routing with convergence guarantee in section 4, axiom formulation of network forwarding in section 6, BGP configuration analysis tool *rcc* in section 5 and Alloy based addressing analysis in section 7.

Incorporated with the evaluations of individual systems presented in previous sections, we now present a comparison and evaluation of the different systems and highlight the ranking of each technique with respect to the corresponding criteria/dimension. As summarized in Table 2, the following criterion and dimensions are considered.

- Application scope dimension specifies the scope of the formal analysis system. For example, metarouting is an example of formal specification system, whereas axiomatic formulation can be used for both specification and verification, and rcc is only used for system validation.

- Expressiveness and automated reasoning support are a pair of conflicting properties that reflect the design choice of the underlying formal analysis system: In theory, the more expressive a specification language is, the less automated reasoning support can be built. In practice, an automatic verification system usually implies the use of a moderate specification language. For example, metarouting and axiomatic formulation represent two extremes in the design of specification language. We will revisit in details the comparison of specification techniques in metarouting and axiomatic formulation in section 8.1.1.

- Soundness and completeness are a pair of dual properties. Soundness ensures that the properties derived in the formal analysis system are indeed preserved in the network specification or system implementation being analyzed. On the other hand, completeness asserts that the analysis of all properties of interests are supported by the analysis system. Obviously, a useful network analysis system should be sound but not necessarily complete since the system may be dedicated to some particular properties. This is observed in the four representative systems studied in this survey: most of them are sound, but only one is complete.

- Closeness to implementation dimension reflects the analysis system's applicability to real-world network protocols. Systems work on unmodified network

systems such as rcc are ranked highest, whereas systems with strong correctness guarantee such as metarouting are ranked lowest.

- Initial investment dimension reflects the additional efforts required in the deployment of the system. Many design choices affect the additional investment required. For example, systems work directly on implementation are also likely to incur only minimal efforts, as in the case of rcc. However, systems that are very flexible and cover a wide range of network properties (i.e. complete), such as axiomatic formulation require very high initial system modeling efforts.

In the rest of this section, we discuss in details the comparison between formal specification, verification, and system validation. We first compare in details different formal specification techniques. We omit the comparison of formal verification techniques, because the full formal verification often restricted to to network standard-/design is of less interests. And though lightweight formal verification has witnessed application in networking recently, they suffer similar scalability problem: high-initial investment in specification and non-incremental verification that are hard to reuse and scale. Instead, we compare formal verification with system validation techniques.

8.1.1 Comparison of formal specification techniques

We have discussed two types of formal specification formalism: metarouting as an example of network meta-theory, and axiomatic formulation as an example of axiomatic semantics framework. They each represents an extreme in the design of specification language with regard to the expressiveness power and automated reasoning support.

Expressiveness power As a specialized meta-model dedicated to BGP routing and convergence property, metarouting formalism is restricted to a particular type of routing policy that is expressed along the traversed path. Metarouting specification language features a set of pre-defined base algebras and composition operations. The human user is left with little specification choice: the user simply focuses on high-level policy decomposition and figures out a mapping from the desired conceptual policy composition to an algebraic composition expressible in metarouting algebras. In contrast, the axiomatic formulation is flexible by providing a minimal set of primitive operations. When combined with normal sequential control statement, the the operations with semantics given in Hoare-logic serve as a general-purpose programming language. System behaviors and architectural invariants of the Internet can be formally specified.

Automated reasoning support Correctness properties (convergence) is automatically enabled in metarouting at the expense of limited specification expressiveness power. Metarouting completely removes the verification efforts required to establish correctness. That is, metarouting is an example of "correctness by construction" specification language. In the other extreme, axiomatic formulation framework comes with no proof strategy. The only verification support is the rigorous defined primitive operation semantics in Hoare-style logic that serves as a network-specific proof system according to which deductive reasoning can be performed.

8.1.2 Comparison of system validation and formal verification

Practical network system validation systems today are based on either static constraints checking or model checking. Compared with the traditional general-purpose full-formal verification approach, system validation systems suffer the following problems: (1) the checkable properties are limited to specific constraints or temporal properties; (2) as best-effort verification system built-upon "common belief" in practice, the verification results are often neither sound nor complete. Both false positive (correctness property violations that will not cause problem in real system) and false negative (error miss) are present.

On the other hand, system validation requires significantly less human investment. For example, validation systems of both types [22, 16, 6] are performed over unmodified real-world implementation. *rcc* in particular, even takes care of vendor-specific configuration language. At the same time, the difficult full-formal deductive reasoning process is also removed and replaced by fully-automatic logical evaluation or algorithmic proof search.

8.2 Challenges

Despite the long-history of formal methods application in both hardware and software verification, serious formal network analysis is just beginning to emerge. The revival of formal analysis application in the pragmatic networking domain is encouraged by the growing network complexity that has becoming less manageable with the present performance-centered bottom-up engineering approach. However, the various problems the systems discussed in this survey suffer also reflect the difficulty in the application of formal methods in real-world distributed systems. Regarding the scale and heterogeneity of today's Internet, the emerging formal network analysis is facing enormous challenges. We discuss only a few network-specific challenges here.

Performance analysis and network dynamics Most formal analysis techniques are restricted to logical correctness. This is largely due to the difficulty in formalizing any performance related issues. However, performance related properties play a key role in network correctness. For example, performance properties are often part of the network service specification. One of the obstacles in formalizing network performance is the lack of formal model for network dynamics and distributed time (state): (1) classical logic-based reasoning does not have a built-in notion for time or state; (2) state-aware techniques such as model checking suffers state-explosion problem, and easily blows up with respect to the inherent huge searching space in networking.

Distributed/Local analysis of global network-wide properties Many interesting correctness properties are global and network-wide. However, it is likely that formal analysis can only be applied locally at a restricted sub-network even if the properties of interests concerns external network nodes. This is because the global Internet today is partitioned into a set of administrative domains where each network operator is only granted access to his/her own domain.

A second difficulty has its root in the distributed nature of almost all interesting network application. Distributed algorithm analysis is much harder compared with a centralized one, let alone the formal distributed system verification.

Unified framework for heterogeneous verification system The global Internet is built upon heterogeneous sub-networks, and is multiplexed for many different types of services. As a result, each end node implements a stack of protocols to handle some particular aspects of Internet service. Similarly, most formal analysis systems are specialized for a particular type of correctness properties, and are good for specific application domains. Internet utilizes layering as the abstraction and modularization mechanism to build an integrated service out of a set of protocols. However, the construction of correctness proof of a network service from the correctness proofs of the constituent component protocols is still an open problem. A related issue is the formal argument reuse problem that hinders the scalability of formal analysis.

9 Acknowledgments

I would like to thank Prof. Rajeev Alur for chairing my WPE-II committee, as well as Prof. Jonathan M. Smith and Jennifer Rexford for sitting on the committee.

References

[1] Isabelle proof assistant. http://isabelle.in.tum.de/.

[2] The Coq Proof Assistant. http://coq.inria.fr.

[3] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 1986.

[5] N. Feamster and H. Balakrishnan. Towards a Logic for Wide-Area Internet Routing. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture 2003*, Karlsruhe, Germany, August 2003.

[6] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *2nd Symp. on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[7] A. P. Felty, D. J. Howe, and F. A. Stomp. Protocol verification in nuprl. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 428–439, London, UK, 1998. Springer-Verlag.

[8] L. Gao and J. Rexford. Stable internet routing without global coordination. *SIGMETRICS Perform. Eval. Rev.*, 28(1):307–317, 2000.

[9] T. G. Griffin and J. L. Sobrinho. Metarouting. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 1–12, New York, NY, USA, 2005. ACM.

[10] T. G. Griffin and G. Wilfong. An analysis of bgp convergence properties. *SIGCOMM Comput. Commun. Rev.*, 29(4):277–288, 1999.

[11] A. J. T. Gurney and T. G. Griffin. Lexicographic products in metarouting. *Network Protocols, IEEE International Conference on*, 0:113–122, 2007.

[12] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing extensible ip router software. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 189–202, Berkeley, CA, USA, 2005. USENIX Association.

[13] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 26(1):53–56, 1983.

[14] D. Jackson. Software abstractions: Resources and additional materials.

[15] M. Karsten, S. Keshav, S. Prasad, and M. Beg. An axiomatic basis for communication. *SIGCOMM Comput. Commun. Rev.*, 37(4):217–228, 2007.

[16] C. E. Killian, J. W. Anderson, R. Braud, R. Jhala, and A. M. Vahdat. Mace: language support for building distributed systems. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 179–188, New York, NY, USA, 2007. ACM.

[17] C. E. Killian, J. W. Anderson, R. Jhala, and A. Vahdat. Life, death, and the critical transition: Finding liveness bugs in systems code (awarded best paper). In *NSDI*. USENIX, 2007.

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM TOCS*, 18(3):263–297, 2000.

[19] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: language, execution and optimization. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 97–108, New York, NY, USA, 2006. ACM.

[20] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, 39(5):75–90, 2005.

[21] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 289–300, New York, NY, USA, 2005. ACM.

[22] M. Musuvathi, D. Y. W. Park, A. Chou, D. R. Engler, and D. L. Dill. Cmc: a pragmatic approach to model checking real code. *SIGOPS Oper. Syst. Rev.*, 36(SI):75–88, 2002.

[23] PVS Specification and Verification System. `http://pvs.csl.sri.com/`.

[24] A. Rodriguez, C. Killian, S. Bhat, D. Kostić, and A. Vahdat. Macedon: methodology for automatically creating, evaluating, and designing overlay networks. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association.

[25] J. L. Sobrinho. Network routing with path vector protocols: theory and applications. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 49–60, New York, NY, USA, 2003. ACM.

[26] A. Voellmy and P. Hudak. Nettle: A language for configuring routing networks. In *DSL '09: Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*, pages 211–235, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 15–15, Berkeley, CA, USA, 2004. USENIX Association.

[28] Yices: An SMT Solver. `http://yices.csl.sri.com/`.

[29] Z3. http://research.microsoft.com/projects/Z3/.

[30] P. Zave. A formal model of addressing for interoperating networks. In J. Fitzgerald, I. J. Hayes, and A. Tarlecki, editors, *FM*, volume 3582 of *Lecture Notes in Computer Science*, pages 318–333. Springer, 2005.