Network Policy Languages: A Survey and a New Approach

Gary N. Stone, Bert Lundy, and Geoffrey G. Xie, U.S Department of Defense

Abstract

In this article a survey of current network policy languages is presented. Next, a summary of the techniques for detecting policy conflicts is given. Finally, a new language, Path-Based Policy Language, which offers improvements to these is introduced. Previous network policy languages vary from the very specific, using packet filters at the bit level, to the more abstract where concepts are represented, with implementation details left up to individual network devices. As background information a policy framework model and policy-based routing protocols are discussed. PPL's path-based approach for representing network policies is advantageous in that quality of service and security policies can be associated with an explicit path through the network. This assignment of policies to network flows aids in new initiatives such as integrated services. The more stringent requirement of supporting path-based policies can easily be relaxed with the use of wild card characters to also support differentiated services and best-effort service, which is provided by the Internet today.

Millions of dollars are lost, a company folds, and thousands of employees are let go, throwing a community into economic chaos. These events could result from discontinued funding from Silicon Valley investors becoming aware of continued reports of poor quality of service (QoS), security problems, and the inability of clients to access the company's Web sites. This ingenious company was the first to integrate voice, video, and data all on the same network based on a company's policies. The company's untimely demise was the result of *conflicting network policies* that were disseminated automatically throughout the network's policy servers, causing erratic network performance. This scenario, although extreme, shows the importance of being able to represent the network policy goals of a company while simultaneously verifying that those goals do not conflict with each other.

To understand how policy can play a role in managing a network, policy must be defined and applied to communications networks. The Internet Engineering Task Force (IETF) has proposed an Internet-draft of terminology for describing network policy [1] and provides many of the definitions used throughout this article.

A policy is formally defined as an aggregation of policy rules [1]. Each policy rule is made up of a set of conditions and a corresponding set of actions. The conditions define when the policy rule is applicable. Once a policy rule is so activated, one or more actions contained by that policy rule may then be executed. These actions are associated with either meeting or not meeting the set of conditions specified in the policy rule. In other words, a policy specifies what action(s) must be taken when a set of associated conditions are met.

A simple view of policy in regard to networks is that policy constrains communications. Specifically, network policy defines the relationship between clients using network resources and the network elements that provide those resources. A *client* in this case refers to users as well as applications and services.

Network policy allows administrators to manage network elements to provide service to a set of clients. If every system were permitted to communicate with all other systems without restriction, there would be no need for network policies. Increasingly, networks that once only supported best-effort traffic are integrating voice and data as well. Without a means for network managers to control the use of the network, mission-critical applications and general network performance are going to suffer, and there will be little hope of supporting future real-time applications.

Network policies are grouped into three general areas:

- How the policy is used
- How the policy is triggered
- At which level the policy is applied

A *usage policy* describes *which* services will be used to

This work was supported in part by DARPA under the NGI program and a grant from NASA Ames Research Center.

 U.S. Government Work Not Protected by U.S. Copyright

 SAMSUNG 1036

maintain the current state of the network or to transition to a new state. Services which may be available in the network are differentiated service classes, virtual private networks, encryption capability, and so on. A usage policy also describes *how* those services will be used. For example, the ability to differentiate the handling of separate flows of traffic based on the service class in which they reside or to which virtual channel they belong describes how a service is used.

Policies can be *triggered* in two ways, either statically or dynamically. "Static policies apply a fixed set of actions in a pre-determined way according to a set of pre-defined parameters that determine how the policy is used" [1]. Examples of static policies are: transit traffic is not permitted during normal working hours; Internet radio is only permitted after 4:00 p.m.; for security reasons certain network addresses are denied access to network resources.

Dynamic policies are only enforced when needed, and are based on changing conditions of the network such as congestion, packet loss, or the loss of a network router. To support the dynamic and sometimes unexpected nature of the network, actions can be triggered when an event causes a policy condition to be met. Examples of dynamic polices are: when the network gets congested, streaming video traffic is disallowed; or when a particular service class of user is utilizing the network, lower best-effort traffic to only 25 percent of link capacity.

Last, the *level of the policy* is applied as a category. These policies are differentiated by their granularity, such as the application level, user level, class level, or service level. For example, a mission-critical application may be given priority over all other network traffic, or all users in the silver class (differentiated services) have priority over the bronze class, but must succumb to the gold class.

Later in this article the authors introduce a policy language based on path. *Path-based* policy is defined as a policy were all attributes associated with the policy, which include the service type of the traffic, conditions used to trigger the policy, and the actions executed when the policy is triggered, are all bound to a predefined path. Using the path as the fundamental building block of a policy statement provides great control and flexibility. The ability to specify an explicit path, which represents each node from source to destination, enables us to create virtual channels where resources are reserved to support real-time applications. These paths can be specified by either a user or a network administrator. If a path had the restriction of *always* including each node in the path, the number of unique paths needed to support a network could soon become overwhelming. This is why a path may include wild card characters, and thus adds great flexibility to the way policies are specified. The use of a wild card character allows for path aggregation, which greatly reduces the number of paths that have to be specified, and at the extreme one path statement can specify all possible paths under an administrator's control.

Policy-based networking — the ability to control a networking environment by specifying and enforcing policies — is gaining increased interest among the network community. Policy-based networking helps manage user and application priority, QoS, and security rights based on management policies. Because of an increasing industry trend to deploy business applications over the network and the convergence of voice, video, and data applications on the same network, major network vendors such as Cisco, Nortel, and Lucent Technologies are developing products to support network management. These products allow network managers to create and implement policies that can prioritize the use of network resources by different network applications so that bandwidth will be guaranteed to the most business-critical applications during times of network congestion. For example, a company that offers IP telephony — which has strict timing requirements — must not permit a large data file transfer to interfere. Network management also provides the ability to restrict the use of network segments by denying access to unwanted and perhaps malicious traffic. The ability to create and enforce network policies adds intelligence to a network that was previously based only on best-effort packet traffic. Rather than adding more bandwidth, which is expensive and time-consuming, to solve existing network congestion, companies can use network policies to allow for important applications and user groups to receive network priority over secondary network users. It is important to have an underlying mechanism to support and enforce these defined network policies. The IETF has attempted to define such an architecture in the working draft, "Policy Framework Core Information Model" [2].

Many of the aspects of this mechanism needed to support policy-based networking are being addressed, such as policy storage structures, policy servers, and protocols to deliver translated policies to enforcement points. One aspect of policy-based networking that does not seem to be receiving much attention is the verification of policies that are going to be applied to the network. Consistent enforcement of network policies, often specified by different people at different times, is impossible if those policies conflict with each other. Thus, a method is needed to detect and deal with conflicting policies before they are distributed throughout the network to the policy enforcement points.

IETF Policy Framework Core Information Model — With the emergence of service models such as differentiated services (DS) [3, 4], integrated services (IS) [3, 4], and multiprotocol label switching (MPLS) [4], the IETF has published a working draft for terminology to describe network policies and services [2]. The IETF tries to standardize a mechanism as well as a language in this draft by developing a scalable architecture for policy administration and distribution of network policies. A key to this architecture is a common language to represent and provide a consistent implementation of policy.

An underlying assumption of this draft is that policies are stored in a centralized repository. The policy repository is one of three important entities of the model. The other entities are the policy enforcement points (PEPs) and policy decision point (PDP).

A PEP is a component of a network node (e.g., a router, switch, or hub) where the policy decisions are actually enforced. When the PEP requires a policy decision about a new flow of traffic or authentication, for example, the PEP will send a request to a PDP.

The PDP is the entity in the network where policy decisions are made. This PDP, which may reside on a remote server, will make policy decisions using information retrieved from policy repositories.

Communication is needed to and from the policy repository as well as between the PDP and the PEP. In many proposals the policy repository is a directory, and therefore the appropriate access protocol would be the Lightweight Directory Access Protocol (LDAP). Examples of a policy protocol, which is used to request and reply to policy decisions, could be the Common Open Policy Service (COPS) protocol [5] and Simple Network Management Protocol (SNMP) [6].

Since the PEPs can potentially be from multiple vendors, a common policy language is needed to support the dissemination of policy information to these devices. In the Policy Framework Core Information Model [2], policy is defined as

((*Hs*, *ARs*, *ARent*), (*Hd*, *ARd*, *ARexit*), *UCI*, *Cg*)
where:
Hs is the source host address
ARs is the source AR
ARent is the entry AR (previous hop)
Hd is the destination host address
ARd is the destination AR
ARexit is the exit (last hop)
UCI is the User Class Id (e.g., Gold, Silver Bronze service levels)
Cg are any global conditions

■ Figure 1. *A policy term.*

an aggregation of policy rules. Each of these policy rules is composed of a set of conditions and a set of actions to perform if the conditions are met. The general form of these conditional statements is

IF <condition 1> **AND** <condition 2> … **AND** <condition *N*>
THEN <action 1> … **AND** <action *N*>

The policy representation includes a means to prioritize and order both the conditional statements as well as the policy actions. This is crucial when multiple policies exist and conflict. A conflict occurs when the conditions of at least two policies are simultaneously satisfied, but the actions of at least one of the policies cannot be simultaneously executed. For example, a router may have two access control rules where their conditions are simultaneously satisfied, but one contains that action deny, the other permit. For example:

access-list 1 permit 131.1.30.00.0.0.255
access-list 1 deny 131.1.0.00.0.255.255

The first permits traffic with IP addresses beginning with 131.1.30 to pass. The second rule conflicts with the previous one by denying traffic with any IP address beginning with 131.1. The first rule in an access list that satisfies the conditional requirement is executed. This procedure resolves conflicts but puts the onus on the operator to enter the rules in the correct order.

The rest of this article is organized into six sections. We review policy-based routing protocols. A lot of early work on the use of polices in networks occurred in the context of these protocols. We discuss languages used to represent network policies. These languages are at a more abstract level and can be used to describe policies without low-level details. An abstract language is beneficial when multiple devices and vendors are involved. We will discuss languages used to describe network traffic at a low level such as the protocol data unit (PDU) level. These languages are more adept at defining patterns for the selection of network traffic in the conditional section of a policy. This article contains a review of research efforts that involved using formal logic to determine consistency between policies. This formal logic section provides background information for features introduced with our new language. We introduce our language, the Path-Based Policy Language (PPL). Our goal is to represent network policies at an abstract level in order to support heterogeneous networks, while also providing translation of those policies into formal logic. Having policies represented in logic will provide the ability for theorem provers to detect conflicts. We then summarize the article.

Policy-Based Routing Protocols

In this section three policy-based routing protocols are reviewed. They provided a lot of early work on the use of policies in networks. All of these protocols enable policies to be enforced based on the elements of an explicit path through the network.

Border Gateway Protocol

Lougheed and Rekhter define an interautonomous routing protocol, Border Gateway Protocol (BGP) [7–9], where routers share reachability information by passing autonomous system[1] (AS) information between neighbors. This exchange of routing information contains full AS paths the traffic will transit to reach a distant network. Path information is not only useful in removing loops in the network, but also allows policy decisions to be made at the AS level. Policy enforcement is not part of the protocol itself, but instead is manually configured at each BGP router.

Policy decisions made by BGP [10, 11] are based on configuration information manually configured into each router by an AS administrator. The enforcement of policies is accomplished in two ways. The first is by specifying the procedure used by the AS router itself to select the best paths, and the second is by controlling the redistribution of routing information to neighboring ASs.

Policy decisions can be based on various preferences and constraints. Since the complete AS path is advertised to neighboring routers, particular paths can be rejected based on an AS that is contained in the path. The reasons a particular path are rejected vary. For example, a particular AS whose control is under that of a major competitor may want to be avoided, causing one or more paths that include this AS to be eliminated from consideration. Performance information can also be used to eliminate paths from consideration. If an AS has access to metrics related to performance such as link speed, delay or capacity, these measurements can be used to rate multiple paths for selection.

BGP, allowing an AS to control redistribution of routing information, is the means by which an AS can enforce policies on others. For example, if an AS does not want to be used for transit traffic, it does not advertise routes to networks other than those directly connected to it.

Fundamentally, BGP is a distance-vector protocol, but instead of maintaining just the cost to each destination, BGP keeps track of the exact path used. This use of the path categorizes BGP as a path-vector protocol. As mentioned earlier, policies are not part of the BGP protocol itself; therefore, each AS may have its own means of evaluating paths. Each router contains a module for examining paths to a given destination and scoring them. This scoring mechanism, which may include local policy information, is then used to choose the best path to a destination. BGP routers can only advertise paths they themselves use. This prevents an AS from sending datagrams to a distant network using one path, but advertising an alternative path for others to use.

Inter-Domain Routing Protocol (IDRP)

Kunzinger and Thomas describe the Inter-Domain Routing Protocol (IDRP) [12, 13], which is the International Organization for Standardization (ISO) protocol for routing between ASs. IDRP came out of the same work as BGP, and may be considered a follow-on to BGP. Just as in BGP, IDRP supports policy-based routing, but is not concerned with the implementation details of those policies. Policy-based routing can restrict access, and therefore enforce policy, by controlling the distribution of routing information to neighboring routers.

[1] *AS and administrative region (AR) are a set of routers under a single technical administration, using one or more interior gateway protocols to route packets within the AS, and using an exterior gateway protocol to route packets to other ASs [3].*

This selective distribution of information can enable the AS to deny all transit traffic, or deny access to only certain network paths.

The IDRP router accepts router information from neighboring routers, which express their views of the network, and uses this gathered information to construct its own view of the network. The IDRP router at this point can use local policy information to select or reject routes accordingly. The IDRP router advertises its view of the network with internal gateway protocols such as Open Shortest Path First (OSPF) or Routing Information Protocol (RIP) so that all routers within the AS have a consistent view of the network.

Just as ASs were used to refer to an entire set of IP networks, IDRP supports a concept called *routing confederations*. A routing confederation is a grouping of ASs to make managing the Internet more manageable. As the Internet has grown, the number of ASs has also grown, making its management less efficient. These routing confederations are quite flexible in that they can be subsets of and even overlap each other.

IDRP uses path-vector routing to propagate routing information. Path-vector routing, as used in BGP, explicitly lists the entire path to each destination. This concept can alleviate network loops as well as enforce policy constraints based on the ASs or confederations that make up the path.

Another feature supported in IDRP is the ability to reduce the number of path vectors by using route aggregation. Route aggregation lets an IDRP router combine multiple IP address prefixes or destinations to create a single advertisement for them all. This feature greatly reduces the number of individual destinations a router must support as well as reducing the amount of data that has to be sent during the advertising phase.

Inter-Domain Policy Routing

Steenstrup presents a set of protocols [14] and architecture in [15] for Inter-Doman Policy Routing (IDPR). Unlike BGP and IDRP, IDPR uses link state routing to provide policy routing among administrative domains (ADs).[2] The primary objective of IDPR is to provide traffic with routes that satisfy the users' service requirements while respecting the service providers' service restrictions [16]. *Source policies* represent the users' requirements and can consist of parameters such as throughput, acceptable delay, cost of session, and domains to avoid. Service providers specify *transit policies*, which specify offered services and the conditions of their use.

The generation and selection of policy routes is based on distributed routing information and the source policies specified by the domain administrator. IDPR forwards messages across paths established using the policy routes generated. Route generation is inherently complex and the most computationally intensive part of IDPR. The general policy route generation problem involves a combination of service constraints, for example, finding a route delay of no more than *S* seconds and a cost no greater than *C*. Most of these multiconstraint routing problems are NP-Complete.

To reduce the size of the link state database, IDPR supports the ability to group ADs into superdomains. The existence of superdomains imposes a domain hierarchy within the network. With a hierarchical approach only domain-level information is needed to construct routes. This greatly reduces the information needed to be maintained by a route server. The size of the database will now depend on the number of domains and the policies associated with each.

A variant of Clark's policy term was chosen to represent policies in [15]. This variant allows for policies to be associated with a set of network elements that represents a path. A policy based on paths is a great asset to policy-based routing protocols.

[2] *Administrative domain refers to any collection of contiguous networks, gateways, links, and hosts governed by a single administrative authority who selects the intradomain routing procedures and addressing schemes, specifies service restrictions for transit traffic, and defines service requirements for locally generated traffic.*

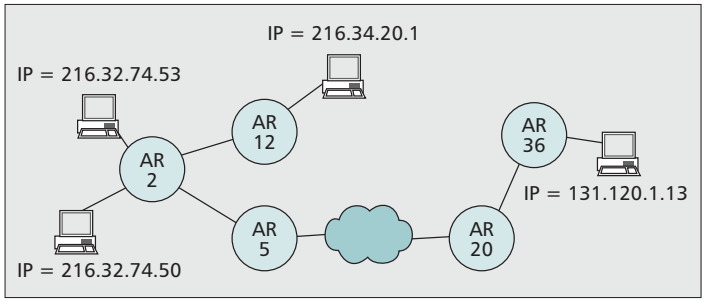


Figure 2. *A sample network diagram.*

Network Policy Languages

In this section all the major policy languages are discussed. These languages are used to represent varying types of network policies such as routing, access, and QoS.

Clark's Policy Term

Seeing the importance of using network resources differently and more efficiently, Clark proposed a template to represent network policies [17]. This template, called a *policy term*, was designed to enable a wide range of network policies to be represented. The work is based on the fundamental assumption that Internet resources are grouped into administrative regions (ARs). AR resources included such items as networks, links, routers, and gateways. The format of a policy term is shown in Fig. 1.

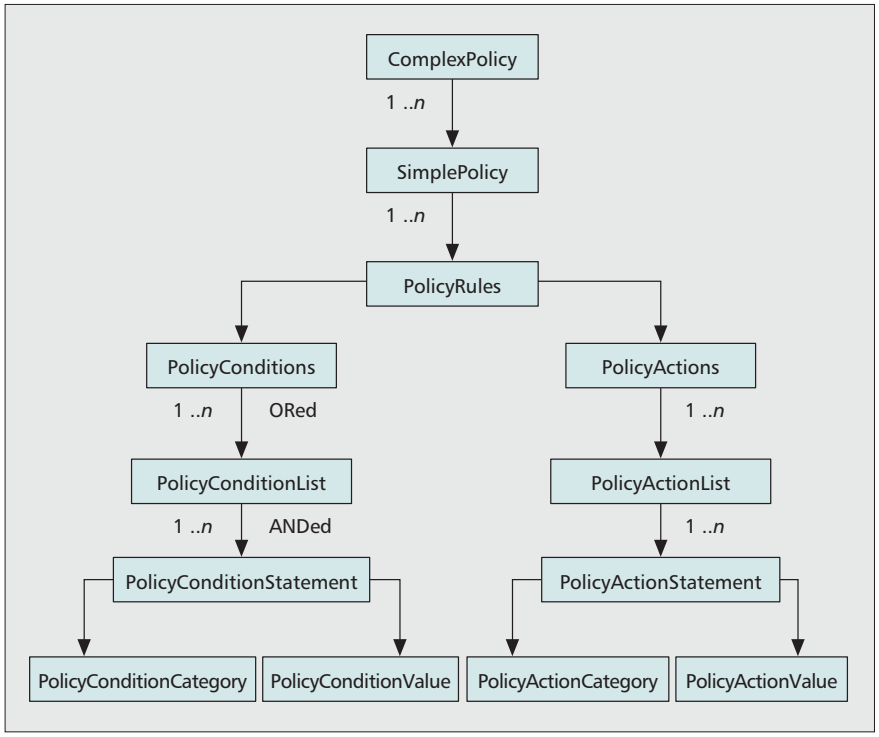
The first two "elements" of the policy term represent the source and destination points, respectively. Each of these consists of three parts, which provide for a wide range of granularity while specifying the endpoints. To show the granularity available with this schema, here are some examples of source and destination points that could be represented by the diagram in Fig. 2. These source and destination points use the special characters "*" and "-." The "*" represents the wild-card match, and the "-" is used to make sure the AR entry or AR exit fields match the source AR or destination AR, respectively. These examples could be applied to AR 2.

(*, *, *) (*,*,*)
No restrictions, allow all traffic flows to traverse without restriction.

(*, 36, -) (*, 12,*)
Allow all hosts directly attached to AR 36 to pass if their destination goes through AR 12 (e.g., host 131.120.1.13 may communicate with 216.34.20.1).

(131.120.1.13, 36, -) (216.32.74.53, 2, -)
The host with IP address 131.120.1.13 in AR 36 may communicate with the host with IP address 216.32.74.53 in AR 2.

As the reader can see, the endpoints can be as explicit as specifying a host or generic enough to allow all Internet traffic. Although the use of these first two elements pro-



■ Figure 3. *A PFDL hierarchy.*

vides for a flexible way to permit traffic flow, it can become cumbersome at times. If, for example, the reader wanted to allow only traffic from universities to flow across an AR, it could be accomplished by creating many policy terms, one for each university. This list of policy terms could become quite large, so the third "element" of the policy term, UCI, can be used to make the implementation of this policy more manageable. A policy term that would only allow university traffic to flow across an AR could be represented like this:

((*,*,*), (*,*,*), University, *)
The endpoints are such that if the UCI element was not used, all traffic would flow across the AR. Using the UCI as a filter, only traffic marked with a **University** tag would be permitted to pass.

The last element field of the policy term, Cg, is used for global conditions. Examples of information that might be held in this field are, "unauthenticated UCI," "no-per-packet charge," and "limited to *n*% of available bandwidth."

((*,*,*),(*,*,*), University, {unauthenticated UCI})
This would allow only traffic marked as **University** to flow through the AR. There is no need to verify that the packet traffic was really from a university host.

Although this was a good start for an abstract network policy representation, which is needed for heterogeneous environments, it has limitations. There is no ability to represent explicit paths formed by a sequence of ARs as part of the term. Only single ARs and the wild card character "*" are allowed. Without this capability, support of Integrated Services requires several network policies distributed throughout the network to be combined for verification of a path. Consequently, there is no ability to exclude a set of ARs from a term to which a general policy is applied.

Policy Framework Definition Language

Strassner and Schliemier define the Policy Framework Definition Language (PFDL) [18] which provides a mapping of network service requirements from a business specification to a vendor- and device-independent format. The benefit of such a language is that network policy can exist in a heterogeneous environment of devices that support policy enforcement.

With the development of standards to provide QoS, like integrated services with Resource Reservation Protocol (RSVP)[3] and differentiated services, the IETF working group on Policy Management has proposed this language. The belief is that without a means of representing, administering, and distributing consistent policy information, these QoS standards that classify and give preferential treatment to certain types of traffic flows will not see widescale deployment.

In this first release of the draft, the grammar was only available in Backus-Naur Form (BNF), and no explicit examples were presented. Attributes the authors believe should be supported by the language are discussed. As with many of these efforts to represent policy, the authors believe that having a language which supports multiple network devices and vendors is the key to successful policy deployment.

The design of PFDL is based on the Common Information Model (CIM) [2] being designed by the Distributed Management Task Force (DMTF). This model defines a hierarchy of object classes that can be used to represent policy information.

The class and relationship hierarchy of the CIM model are used to help define the structure of the PFDL grammar, see Fig. 3. The basic premise is that a policy is an aggregation[4] of policy rules. A policy rule defines a sequence of actions to be initiated when a corresponding set of conditions is satisfied. Five classes defined to support the CIM are the *ComplexPolicy* class, *SimplePolicy* class, *PolicyRule* class, *PolicyCondition* class, and *PolicyAction* class. Their relationship to each other is shown in Fig. 3.

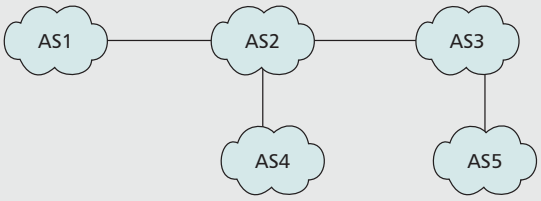
A *PolicyRule* contains a set of *PolicyConditions* and a set of *PolicyActions*. When the set of *PolicyConditions* are meet, the set of *PolicyActions* will be executed.

A *PolicyConditionStatement* is composed of a category and value pair. These two components are specific to a particular knowledge domain, whether QoS, security, or any other domain. Providing conditions and actions for a given knowledge domain accommodates the interoperability requirement for the language. It will provide the means for multiple vendors to supply components to a general policy architecture.

A *PolicyAction* is a class in the PFDL model that consists of

[3] *RSVP defines how applications can place reservations, and how they can relinquish those resources once their need ends.*

[4] *An aggregation is a string form of an association. An aggregation is usually used to represent a "whole-part" relationship.*




```
aut-num:     AS2
as-name:     CAT-NET
deser:       Catatonic State University
import:      from AS1 accept ANY
import:      from AS3 accept <^AS3+$>
export:      to AS3 announce ANY
export:      to AS1 announce AS2 AS3
admin-c:     AO36-RIPE
tech-c:      CO19-RIPE
mnt-by:      OPS4-RIPE
tech-c:      CO19-RIPE
mnt-by:      OPS4-RIPE
changed:     orange@ripe.net
source:      RIPE
```

■ Figure 4. *An RPSL diagram and policy example.*

an action or a list of actions that will be executed when the conditions associated with a policy are evaluated to true. These actions can be executed in either a specific order or any order, which is the default. Along with the ordering of policy actions, the ability exists for the conditional execution of one or more actions based on the results of previous actions. The reader can see from Fig. 3 that the hierarchy of the PolicyActions class is similar to the *PolicyConditions* class.

With possibly hundreds, perhaps thousands, of policies to be supported in a network, the ability to detect conflicting policies is crucial. The authors of PFDL are aware of the need to both detect as well as support facilities to resolve conflicts. This proposal groups policy conflicts into two different categories, intra- and interpolicy conflicts.

Intrapolicy conflicts are caused when the conditions of at least two policies are simultaneously satisfied, but the execution of the actions of these policies cannot be executed at the same time. Interpolicy conflicts are described as two or more policies that, when applied to the network, result in conflicting configuration commands specified for one or more network devices. In this case, the conflict exists when the policy is applied to a specific network or device(s). An example given in the proposal is when two policies are executed such that the number of queues in one network device is such that it does not match the number of queues allocated in a second device supporting the same traffic flow.

Once conflicting policies are detected, they may be resolved in several different ways. The most obvious would be to modify the conditions or actions of the policies to remove the conflict. If this cannot be accomplished and the conflicting policies must exist in the system, there are three different ways to resolve them:

- Resolve the conflict by only executing the first policy in the conflicting set.
- Use a priority scheme where only the highest-priority policy in a conflicting situation will be executed.
- Use some type of metadata to determine which rule should be applied. The difference between this and straight priority is that priority is inherently linear, whereas metadata enables nonlinear solutions, such as branching, to be used.

PFDL does not support path-based policies. A path-based capability aids in initiatives such as integrated services [3, 4] and server and agent-based active network management (SAAM) [19]. PFDL is a nice high-level framework, but lots of details need to be filled in.

Routing Policy Specification Language

One of the activities of the Routing Policy System working group of the IETF is to develop a language for describing routing policy constraints. Alaettinolgu, Meyer, *et al.* provide a reference for the language [20] and a guide on how to use the language [21]. Routing Policy Specification Language (RPSL) is a replacement for RIPE-81, the first language deployed in the Internet for specifying routing policies, and is the current Internet policy specification language. Specifying policies in RPSL allows a network operator to specify routing policies in the Internet routing registry (IRR) so that policies and announcements can be checked for consistency. The IRR stores the object-oriented policies of authorized organizations so that others using the whois[5] service can query them. Each object that contributes to a policy stores pieces of information regarding the policy. Each object used to represent the policies contains attributes referred to as *keys* that can be either mandatory or optional. RPSL is designed so that router configurations can be generated from the policies described with the language.

Figure 4 is an example from [21] that represents a common but perhaps simple policy. The *aut-num* represents the AS number; in this case AS2 represents autonomous system 2. The *as-name* and *descr* attributes are the AS's name and description, respectively. The most important attributes of this aut-num are the import and export policies. The import clause specifies import policies, the export clause export policies.

In this example, the import policy of "from AS1 accept ANY" indicates that AS2 will accept any announcements that AS1 sends. The second import policy states that AS2 only accepts announcements from AS3 that originated in AS3 and have paths composed of only AS3s.

The export policy of "to AS3 announce ANY" indicates that any route AS2 has in its routing table will be passed on to AS3. The second export allows the announcements of all routes from AS2 or routes learned from AS3 to be sent to AS1.

The *admin-c* (administrative), *tech-c* (technical), *mnt-by* (maintained by), and *changed* (last changed by) are attributes that contain contact information. The values assigned to these attributes are handles that uniquely identity the person responsible for the attribute. The *source* entry indicates that this object belongs to the RIPE[6] registry.

RPSL represents routing policies well, but was not intended to support policies regarding QoS or general access control mechanisms.

Traffic Flow Languages

This section discusses languages used in the selection of network traffic in the conditional section of a policy. At a lower level than the languages above, these languages can be used for pattern matching in network devices.

PAX Pattern Description Language

Nossik, Welfeld, and Richardson describe PAX [22], a special-purpose language used to define pattern-matching criteria in policy-based networking devices. PAX was intended primarily for data communications networks, but is also generic enough

[5] *WHOIS is used to look up records in a Whois database. Each record has a "handle", a unique identifier assigned to it by the Network Information Center (NIC). Each whois record will also have a name, record type, and various other fields of information, all depending on the type of whois record.*

[6] *The RIPE Network Coordination Centre acts as the regional Internet registry (RIR) for Europe and surrounding areas.*

```
{
        version UINT 4 = = 4;                   /* IP version 4 packet */
        ihl UINT 4 = = 5;                       /* length == 5 : no options */
        typeOfService UINT 8;
        totalLength UINT 16;
        identification UINT 16;
        flagReserved BIT 1 = = 0;
        flagDontFragment BIT 1;
        flagMoreFragments BIT 1 = = 0;          /* last fragments only */
        fragmentOffset UINT 13 = = 0;           /* first fragments only */
        timeToLive UINT 8;
        protocol BIT 8 = = 6;                   /* Next protocol TCP */
        headerChecksum BIT 16;
        sourceAddress BIT 32;
        destinationAddress BIT 32;
}
```

■ Figure 5. *A PAX pattern to match an IPv4 header.*

to be used for any kind of pattern recognition.

The language itself was designed to be much like the C programming language. Viewing the code of a PAX program, the reader will see features similar to C such as comments, preprocessing directives, source file inclusion, conditional compilation, import and export statements, and the use of defines and macros.

The basic concept in PAX is the pattern, with simple patterns combined to form more complex patterns. The use of field concatenation and field combination, and the ability to name patterns leads to a flexible and powerful language for describing patterns in data communication.

Examples from [22] will provide the reader with a quick idea of the syntax and features of the language. Two built-in fields used in the following examples are BIT and UINT, used to create the simplest of patterns.

BIT 16 — matches any 16 bits in the input
UINT 4 — matches any 4 bits; the value of those bits is converted to an unsigned numeric field

Figure 5 illustrates a pattern to match IPv4 headers of TCP/IP nonfragmented packets without IP options. This figure shows how simple patterns can be concatenated together to form more complex patterns. This pattern matches the input only when the 4-bit "version" element is equal to decimal 4, the next element "ihl" equals the decimal value 5, and the subsequent simple patterns are all successfully matched.

Figure 6 represents two more features of the PAX language. The first is the ability to name the pattern for inclusion in other more complex patterns, in this case IEEE_802_2_LLC. The second is the use of a conditional field. Conditional fields are used to describe patterns with varying layouts depending on previous fields. Here when the Control1 field is equal to 0b11, the next 6 bits are used to create a field called ShortControl. When the value of Control1 is not equal to 0b11, the next 14 bits are used to create a field called LongControl.

Simple Ruleset Language

Brownlee describes the Simple Ruleset Language (SRL) [23] as a procedural language for creating rulesets for real-time traffic flow measurement (RTFM). These rulesets, which specify the flows to be measured and how much information should be collected for each, are downloaded to RTFM meters. The RTFM meters use a pattern-matching engine to match the downloaded rulesets against attributes extracted from traffic flows to select which flows to monitor. The attributes applied to the traffic flows are specific to network traffic, and map to such things as source and destination addresses and port numbers. SRL is not restricted to just traffic metering, but can be useful in any application that involves selecting traffic flows from a stream of packets.

There are two goals of SRL rulesets: to identify network packets that are a part of the flow of interest, and to take some action as a result of the match. The identification of packets is done using IF statements. Actions available include the ability to save flow identification attributes, and to keep statistical data about the attributes saved.

Figure 7 is an example that counts only TCP/IP packets where the destination port is telnet, while saving the source and destination address pair for each packet.

A Summary of Network Policy Languages — Table 1 summarizes the languages from previous sections used to represent network policies or support mechanisms for enforcing policies. The columns of this table represent criteria believed to be useful in comparing the various languages. The "Support automated conflict detection" column refers to the ability to recognize different types of policy conflict as well as to provide a flexible means to resolve conflicts. The column labeled "Suitable for integrated services" takes into account the ability to efficiently support integrated services. An entry with a medium value signifies that the language can represent a path through the network, but multiple polices have to be combined to do so. If a column has a high value, the policy language can represent a path in a direct and intuitive manner, and a policy can be applied directly to that path. The benefits

```
PATTERN IEEE_802_2_LLC {
        DSAP BIT 8 <> 0xFF;             /* Destination SAP not broadcast */
        SSAP BIT 8 <> 0xFF;             /* Source SAP not broadcast */
        Control1 BIT 2;
        LongControl BIT 14 WHEN Control1 <> 0b11;
        ShortControl BIT 6 WHEN Control1 = = 0b11;
}
```

■ Figure 6. *A PAX pattern with a conditional field.*

```
#
# Classify IP port numbers
#
  define Ipv4 = 1;          # Address Family number from RFC 1700
  define telnet = 23;       # Well-known Port numbers from RFC 1700
  define tcp = 6;           # Protocol numbers from RCF 1700
#
  if SourcePeerType == Ipv4 save;
  else ignore;              # Not an Ipv4 packet
#
  if (SourceTransType == tcp && DestTransAddress == telnet)
  save, store FlowKind := 'T';
#
  save SourcePeerAddress /32;
  save DestPeerAddress /32;
  count
#
```

■ Figure 7. *An SRL ruleset to identify and count telnet packets.*

Language	Support automated conflict detection	Suitable for integrated services	Suitable for access control	Target architecture	Ease of representing network policies
Policy term	Low	Medium	High	Distributed	High
PFDL	Medium	Medium	High	Centralized	High
RPSL	Low	Low	High	Centralized	Medium
PAX	Low	Low	High	Distributed	Low
SRL	Low	Low	High	Distributed	Low

■ Table 1. *A summary of languages that represent or support network policies.*

of a high value are that policies which must be associated with all the nodes along a path can be represented with just one statement. This greatly reduces the number of policies statements a domain must maintain. The "Suitable for access control" column refers to the ability to permit or deny access based on policy. The capacity to establish a path through a network and restrict access to that path is of great importance from a security point of view. The column labeled "Target architecture" refers to the storage location of polices. A distributed value means that policies are stored throughout the network, perhaps on individual devices. A centralized value refers to one or only a few locations where all the policies are located. Having a centralized location is beneficial when trying to detect conflicting policies. The last column, "Ease of representing network policies," takes into account the ability of a user to intuitively represent a policy with the language. Targeting our language to the group of individuals responsible for representing policies defined with natural language and entering them into a central repository, the authors believe the more abstract and closer to natural language, the easier they will be to understand. Although these individuals may be versed in formal logic representation, it is believed the majority will be more comfortable with an abstract rule-based language. The more abstract the language and closer to a natural language, the higher the value in the column. The greater the number of details that have to be specified, the lower the value assigned.

Although the languages represented in Table 1 contain many features, none of these languages individually contain all the features provided with PPL. Two major features not adequately addressed by any of these languages are the ability to specify a complete path through a network, and the automatic detection of conflicting policies.

Work in Logic Representation of Policies

This section discusses research using formal logic to represent and detect conflicting policies. The policies represented here are more general and involve the use of natural languages, which tend to be ambiguous, to represent policies ranging from resource management to security.

There has been a great deal of research on the topic of formal representation of policies. Much of this research has been in the area of representing security policies, and the more general problem of translating ambiguous natural language policies into some type of formal representation. Representing network policies with an unambiguous language is the key to detecting conflicts. A network policy language has to be flexible enough to represent a wide range of policies, and at the same time formal enough to support automatic translation to logic. Once the policies are in a logical representation, methods already developed from research in this area will provide a means to check the consistency of multiple policies.

Analyzing the Consistency of Security Policies

In [24], the development of a methodology for reasoning about properties of security policies is discussed. Chovly and Cuppens view a security policy as a specific case of regulation, where a regulation defines which actions an agent is permitted, obliged, or forbidden to perform. With this methodology a system is made up of agents that can perform some actions on some objects. In analyzing the consistency of security policies, focus is put on the ability to perform consistency checks (e.g., check for conflicting situations) on the system, and to have the ability to query a regulation to know which norms apply in a given situation.

Formal logic is used to create an unambiguous representation of security polices. According to [24] the advantage of a representation based on formal logic is the ability to precisely define the axioms[7] to reason about a regulation. With policies defined by axioms, tools can now be developed to check the system regulation for consistency.

Rather than associating norms (i.e., permissions, obligations, and prohibitions) with individuals, roles are created with these attributes and individuals associated with these roles. The individual inherits the norms associated with a role when the individual is playing that role. A conflict can only exist when an individual is playing different roles at the same time, because of an assumption in their research that norms within a role are conflict-free.

To resolve conflicts when an individual is playing multiple roles, an ordering is applied when roles are merged. The order represents a priority between them, and the order is assumed to be total.

Tools written in Prolog were developed that checked the consistency of the security policies as well as an algorithm for solving conflicts when an individual is playing different roles at the same time.

On the Axiomatization of Security Policies: Some Tentative Observations About Logic Representation

In [25], Michael *et al.* add an intermediate step to the traditional approach of translating natural language security policies into their axiom representation. Once the policies are in axiom representation, automated reasoning systems are used to detect conflicts. Errors in translation into axiom form can lead to unidentified conflicts, and incorrect proofs when indeed there is a conflict.

An object-oriented approach is introduced to model the security policies using extended entity-relationship (EER) diagrams. The final axioms of the security polices are then derived from the diagrams rather than directly from the natural language representation. The premise was that overall logic rule formulation is simplified in a model-based approach by capturing many of the rules in the structural model.

A case study comparing the two different approaches, model-based and no prestructuring, produced results that appear to support a premise that fewer structuring errors are made with the model-based approach. A limitation of the model-based approach is that potential queries which might reveal conflicting security polices may be prevented.

[7] *A proposition deemed to be self-evident and assumed without proof.*

Policy Hierarchies for Distributed Systems Management

In [26], Moffett and Sloman form a policy hierarchy by refining general high-level policies into a number of more specific management policies. Refining the goals, partitioning the targets the policies affect, or delegating responsibility to another manager can perform this derivation. The main motivation for understanding hierarchical relationships between policies is to determine what is required for the satisfaction of policies. If a high-level policy is defined or changed, it should be possible to decide which lower-level policies must be created or changed.

The goal of policy hierarchy analysis is to determine whether:

- The collected lower-level objectives will completely achieve the higher-level objective they purport to refine.
- There is conflict between the objectives.
- There is an imperatival policy, with a subject, for each objective. An imperatival policy gives an agent the imperative to carry out an action. In most cases this implies obligation.
- There is an authority policy that empowers the subject to achieve the objective. An authority policy provides an agent with the legitimate power to perform an action.

Conflicts in Policy-Based Distributed Systems Management

In [27], policies are used as a means to specify the management behavior of a system without coding the behavior into the manager agents. Lupu and Sloman focus on techniques and tool support for offline policy conflict detection and resolution. Two types of policies, authorization and obligation, are addressed in this research. Authorization policy specifies which activities a manger is permitted or forbidden to perform on a set of target objects. Obligation policies specify which activities a manager must or must not do to a set of target objects and essentially defines the duties of a manager.

Conflicts can arise in a set of policies, but it is not always desirable to eliminate the conflicts by rewriting the policies or changing the membership of the domains to which policies apply. Since automated managers cannot enforce conflicting policies, Lupu and Sloman suggest that a precedence relationship must be established between polices in order to resolve the conflicts. Four types of policy priority are addressed:

- Negative policies always have priority: negative policies take precedence over positive ones.
- The assignment of explicit priorities: policy 1 has priority over policy 2, which has priority over policy 3, and so on.
- Distance between a policy and the managed objects: priority is given to the policy applying to the closer class in an inheritance hierarchy. For example, a computer science (CS) department is a subclass of a university. If a student is in the CS department, policies of the CS department will override those of the university when a conflict exists.
- Specificity related to domain nesting: a particular case of distance between policies, this principle is that a more specific policy (i.e., a policy applying to a subdomain) refers to fewer objects and so overrides more general policies applying to an ancestor domain.

Lupu and Sloman developed a prototype conflict detection tool that currently detects overlaps between policies and optionally applies domain-nesting precedence. The function of the detection tool is analogous to compile-time type checking for a programming language in that it reduces runtime errors and detects specification errors.

A notation is used to represent policies that are precise and can be analyzed for conflicts using automated tools, but it is not based on a well-known logic. In this system an administrator creates and modifies policies using a policy editor. Checks are made for conflicts, and if necessary policies are modified to remove the conflicts.

Sloman has applied the concept of grouping policies by authorization and obligation, which is then interpreted rather than coded into management agents in several other works [28–30].

A Formal Process for Testing the Consistency of Composed Security Policies

In [31], Michael presents a formal process for testing the logical consistency of composed security policies. The introduction of a structural model is made to represent relationships between security policies, and axiomatizes the policies so that relationships constructed in the model are preserved and made explicit in a logic model. This logic model is then used for deductive proofs of policy consistency. Michael states that problems arise in correctly defining, evaluating, and mapping policies onto procedures, and that a structural model reduces these types of gaps.

OTTER, an automated first-order resolution-style theorem prover, is used to detect logical contradictions between the axioms in the logic model.

The PAIR Project: Policy Analysis of Internet Routing

Although not based on formal logic, the PAIR tools [32] developed under the PAIR Project [33] provide a means to troubleshoot routing and policy problems in the Internet. These tools are most useful in conjunction with the Route Server Next Generation (RSng) route server. These route servers can provide a router with its own view of the network by gathering routing information from neighboring routers, using a route selection procedure, and applying policy requirements for that particular router from the Internet routing registry. Once the routing information is processed, it is passed using BGP to each router for its own view.

The PAIR tools allow peers to diagnose their routing and policy problems by comparing prescribed policy (i.e., policy registered in the Internet routing registry, with policy actually being configured in the Internet. The analysis of routing policy provides the ability to find inconsistencies among policies.

You can also use the PAIR tools to:

- Learn how the policy in the Internet routing registry is processed and used to generate router server configuration files
- Troubleshoot global routing problems
- Identify stale or inaccurate data in the Internet routing registry

Path-Based Policy Language

In this section a new language is introduced which is intended to solve and/or alleviate many of those deficiencies, which were discussed in the previous sections. This new language, designed by the authors, is called the Path-Based Policy Language (PPL).

PPL is designed to support policies that can be applied to both the differentiated as well as integrated services models proposed by the IETF. Several goals for our new policy language are listed below:

- Create a path-based representation of policies flexible enough to support both path- and non-path-based traffic flows. For example, providing an absolute path consisting of the links the traffic must take will provide greater control over traffic flows and provide easier support to integrated services. A less specific policy may only need to provide

source and destination nodes in its configuration, or perhaps just the specification that all traffic of, say, file transfers must be forwarded through a specific node acting as a firewall in an edge router.

- Represent network policies in an unambiguous way. This feature will allow us to detect policies in conflict as well as create a stable network environment.
- Be abstract enough to cross device and manufacturer boundaries. Providing a language that is too specific will demand constant updates to our language as well as to software on the vendor's devices.
- Have ability to resolve conflicts between policies. In order for conflicts to be resolved, they first have to be detected. Having conflicting policies without the means to detect and resolve them is probably worse than having no policies at all. An obvious example involves security access policies. Having policies defined to restrict access to network resources can build a false sense of security, when a rogue policy conflicts with existing policies to provide access to those same restricted resources. Believing that the restrictive policies are working may prevent network administrators or security personal from verifying the protection, which can have very serious consequences. To support the detection of policy conflicts, a follow-on process is being developed which will have the ability to translate policies represented in PPL into formal logic. Once the policies are represented in formal logic, a theorem prover can be utilized to detect conflicts.

Figure 8 represents a summary of the constructs of our language. Several examples of possible policies are provided to show the wide range of policies that can be represented in this path-based approach. Using the wild card character of "*," the ability exists to represent explicit paths as well as groups of traffic flows with our language. This flexibility allows us to represent policies based on QoS and at the same time support existing best effort traffic.

In our language, policy conflicts can be resolved using any of three methods:

- User identification can be used to provide priority of policies based on the creator of that policy.
- The action items of the policy can be used to assign a priority to an individual policy.
- The action items of the policy can also be used to declare compromises where the priority of the policy is lowered if certain conditions exist. Example 5 shows a policy that similarly lowered its own allocated bandwidth for the general good of the network.

The capabilities of our language are illustrated through the use of several examples below. Example 1 shows the ability to specify an explicit path for a traffic flow. Examples 2, 4, 5, 7, and 8 use the "*" to specify partial paths for traffic flows. In examples 3 and 6, the use of "*" places no restrictions on the path the traffic may take.

In example 5 a policy is represented that will make a compromise when certain network conditions are met. This compromise feature provides the ability to throttle back network flows for the general good of the network.

Example 1: **Policy 1 <net_manager> @ {<1,2,5>} {class = {faculty}} {*} {priority := 1}**

This is a rule, which states that the path starting at node 1, traversing to node 2, and ending at node 5 will provide high priority for faculty users.

```
policyID<userID>@{paths} {target} {conditions} [{action_items}]
        policyID        –unique policy identification token
        userID          –user ID of policy creator
        paths           –network paths the policy affects
        target          –target class of network traffic
        conditions      –any global conditions (items are AND'ed)
        action_items    –for setting parameters (e.g., policy priority), declaring
                         compromises and explicit deny, etc.
        action_item = [{condition}:] {actions}

 Semantics: policyID created by <userID> dictates that target class of traffic
may use paths only if {conditions} is true after action_items are performed.
```

Figure 8. *A summary of PPL constructs.*

Example 2: **Policy2 <stone> @ {<*,2,*>, <*,4,*>} {*} time >= 1600, time <= 0800}**

This rule states that all traffic will be allowed to traverse through nodes 2 and 4 during nonworking hours. Unless granted by another policy, traffic will not be able to traverse through nodes 2 and 4 during working hours. This is as a result of the default action, which is an explicit deny.

Example 3: **Policy3 <net_manager> @ {*} {*} {hopCount > 19} {deny}**

This is a rule which states that no path in the network will be permitted if it has a hop count greater than 19. This example shows the ability to use explicit deny.

Example 4: **Policy4 <net_manager> @ {<*,5,*> {*} {hostIP = 131.1.*.*}**

All hosts with a network address starting with 131.1 will be permitted to traverse node 5. Having the ability to restrict groups of network addresses as well as individual network addresses is also a part of our language.

Example 5: **Policy5 <xie> @ {<1,*,2,*,5>}**

{traffic_class = {video, voice}, used_bw <= allotted_bw}

{allotted_bw = 50M, loss_rate (data) > 40% : allotted_bw := 40M}

This policy shows the ability for compromise. Voice and video traffic are provided with an allotted bandwidth of 50 Mb/s, but when the network loss rate is greater than 40 percent, a compromise will be made to lower the allotted bandwidth to 40 Mb/s. In this example used_bw and allotted_bw are user-defined variables. The loss_rate() function is implemented as a message passed from a network device to the policy server.

Example 6: **Policy6 <net_manager> @ {*} {traffic_class = {data}} {*} {priority :=10}**

All data traffic will be assigned a priority level of 10. Assume that there are three classes of traffic for this example, voice, video, and data. This allows for providing higher or lower priority to certain classes of traffic. In this case the priority might affect the ordering of packets being dropped from queues in the network during times of congestion.

Example 7: **Policy7 <Betty> @ {<1,*,5>} {traffic_class = {accounting}}**

{day != Friday : priority := 5}

On all paths from node 1 to node 5, accounting class traffic will be lowered to priority 5 unless it is a Friday. In this

policy the action_items field is used with temporal information to influence the priority of a class of traffic. It might make sense to have this feature when departments of a company need more network resources to accomplish their jobs.

Example 8: **Policy8 <net_manager> @ {<1,*,5>} {traffic_class = {student}} {*}**
{userID = Gary : deny}

On all paths from node 1 to node 5, deny access to network traffic from user Gary who is in the student traffic class. This policy shows that our language can provide control at a very small granularity level. In this case the policy affects only a single user in a particular class of network traffic. It could have easily been modified to provide certain times of the days when it was in effect as well.

In this section PPL was briefly introduced. Our language has the ability to represent network policies unambiguously, providing support to heterogeneous networks for which the networks are controlled using explicit policies. Policies required by both path- and non-path-based traffic flows are supported with our language as well as the ability to resolve conflict between policies.

Our current efforts [34] are directed toward fully specifying the grammar of PPL. Examples 1–8 provide examples of our language, but mechanisms have to be defined to support these policies. Providing the ability for user-defined traffic classes is one example. Constructing a compiler that will verify syntax and detect policy conflicts will take our research forward by providing a mechanism for running experiments.

Concluding Remarks

In this article, several policy languages were reviewed to summarize their purpose, strengths, and weaknesses. Three policy-based routing protocols are summarized for their use of policies in networks. In addition, previous work and techniques used in policy conflict resolution and detection using formal logic were also discussed.

These languages fell into two major categories: abstract network policy languages and bit-level traffic flow languages. The first abstract language is known as the Policy Term, defined by David Clark. It is based on the concept that Internet resources are grouped into administrative regions. The Policy Term was designed to provide for the specification of a wide range of network policies to be represented by supplying source and destination ARs. Other abstract languages are PFDL, a hierarchical object language being designed by the IETF and based on the Policy Information Model; and RPSL, the current Internet routing policy specification language. Examples of languages supporting bit-level details are PDL, used for defining pattern matching criteria in policy-based networking devices; and SRL, which specifies the flows to be measured and how much information should be collected for real-time traffic flow measurement.

It is clear that there is a need for policy conflict detection, evident in the fact that many of the languages surveyed have features to resolve conflicts once they are detected. What is lacking is automatic methods for checking network policies that are to be composed together in order to completely satisfy a corporation's policy goals.

One clear method is to use formal logic to represent network policies. Although this method would make conflict detection much easier with the use of existing theorem provers, most network policy implementers are not as comfortable with this representation.

The existing policy languages discussed in this article are more suited to differentiated services than integrated services as a result of an absence of features to support explicit path-based policies.

Our goal is to develop a network policy language that is more suited to what network policy implementers are accustomed to: a rule-based representation more closely associated with a computer programming language. Taking a path-based approach will enable us to establish policies that will be based on paths, like integrated services, as well as non-path-based policies which are more suited to differentiated services. The use of a wild card character enables us to describe policies based on the concepts of differentiated services or best-effort traffic.

Path-based policies not only are a natural fit for integrated services, but will also aid in the scaling problem, which can occur in supporting policies in large networks. A hierarchical view of a network, which is provided with concepts such as autonomous systems and administrative domains, can greatly reduce the information a route server must maintain. One domain can represent multiple hosts, which not only reduces the size of the database at a route server, but also the information that must be passed between routing entities. This grouping of nodes into regions is only one aspect in the scalability problem of enforcing policies in networks; the number of policies associated with a domain is another. Path-based policies can aid in this problem by providing the ability to assign a single policy statement to multiple paths with one statement. For example, path <1, *, 3> specifies all the possible paths from node 1 to node 3. A policy using this path construct can specify network constraints that are applied to multiple paths. This aggregation of paths can reduce the number of policy statements required at a server.

Path-based policies provide a complexity advantage as well. When a policy server associates policies with nodes rather than paths, a valid path must be constructed for each new request. This construction not only uses node connectivity information to build the possible paths, but applies the policy information from each node as well. When a path generation request involves a combination of service constraints, such as acceptable delay and acceptable throughput, this is now an NP-complete problem. Path-based policies associated with a completely instantiated path are analogous to static routes. Rather than calculating a route through the network, a valid route may be specified ahead of time. When a path is specified in advance with the proper policy constraints, it will accelerate the response to a path request. It may be advantageous to have multiple paths prespecified for a route. These paths may be dynamically updated to account for changing network conditions or situations such as time of day. Multiple paths can also support quick rerouting of traffic flows during network disruptions. However, precomputing paths may also incur excessive overhead if not designed appropriately. A mechanism that properly utilizes precomputed paths with the ability to compute a path on demand would be advantageous.

Our current effort is the development of a compiler to translate policies specified in PPL into formal logic. This will provide us with a means to detect conflicting policies using existing theorem provers. This development will allow us to introduce formal logic into network policy management.

Acknowledgments

The authors would like to thank Prof. Bret Michael for his review of this article and his insightful comments. The authors would also like to thank the reviewers for their detailed and very helpful comments and pointers to further references.

References

[1] J. Strassner and E. Ellesson, "Terminology for Describing Network Policy and Services," Internet draft draft-strasner-policy-terms-01.txt, 1998.

[2] J. Strassner, E. Ellesson, and B. Moore, Eds., "Policy Framework Core Information Model," Internet draft draft-ietf-policy-core-schema-02.txt, Feb. 1999.

[3] R. Rajan *et al.*, "Policy Action Classes for Differentiated Services and Integrated Services," IETF, Internet draft draft-rajan-policy-qosschema-01.txt, 5 Apr. 1999.

[4] X. Xiao and L. Ni, "Internet QoS: A Big Picture," *IEEE Network*, vol. 13, no. 2, Mar. 1999, pp. 8–18.

[5] J. Boyle *et al.*, "The COPS (Common Open Policy Service) Protocol," Internet draft draft-ietf-rap-cops-05.txt, Dec. 1998.

[6] J. Case *et al.*, "A Simple Network Management Protocol (SNMP)," IETF Network Working Group RFC 1157, May 1990.

[7] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol (BGP)," IETF Network Working Group RFC 1105, May 1989.

[8] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol (BGP)," IETF Network Working Group RFC 1163, June 1990.

[9] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol 3 (BGP-3)," IETF Network Working Group RFC 1267, Oct. 1991.

[10] K. Lougheed and Y. Rekhter, "Application of the Border Gateway Protocol in the Internet," IETF Network Working Group RFC 1268, Oct. 1991.

[11] J. Honig *et al.*, "Application of the Border Gateway Protocol in the Internet," IETF Network Working Group RFC 1164, June 1990.

[12] C. Kunzinger, "Protocol for the Exchange of Inter-Domain Routing Information among Intermediate Systems to Support Forwarding of ISO 8473," IETF working draft ISO 10747, Apr. 1994.

[13] S. Thomas, *IPng and the TCP/IP Protocols*, Wiley, 1996, pp. 319–50.

[14] M. Steenstrup, "An Architecture for Inter-Domain Policy Routing," IETF Network Working Group RFC 1478, June 1993.

[15] M. Steenstrup, "Inter-Domain Policy Routing Protocol Specification: Version 1," IETF Network Working Group RFC 1479, July 1993.

[16] M. Steenstrup, "IDPR: An Approach to Policy Routing in Large Diverse Internetworks," *J. High Speed Nets.*, 1994, pp. 81–105.

[17] D. Clark, "Policy Routing in Internet Protocols," IETF Network Working Group RFC 1102, May 1989.

[18] J. Strassner and S. Schleimer, "Policy Framework Definition Language," Internet draft draft-ietf-policy-framework-pfdl-00.txt, 17 Nov. 1998.

[19] G. G. Xie *et al.*, "SAAM: An Integrated Network Architecture for Integrated Services," *Proc. 6th IEEE/IFIP Int'l. Wksp. QoS*, Napa, CA, 1998.

[20] C. Alaettinoglu *et al.*, "Routing Policy Specification Language (RPSL)," Internet draft draft-ietf-rps-rpsl-v2-03.txt, Apr. 6, 1999.

[21] D. Meyer *et al.*, "Using RPSL in Practice," IETF Network Working Group RFC 2650, Aug. 1999.

[22] M. Nossik, F. Welfeld, and M. Richardson, "PAX PDL — A Non-Procedural Packet Description Language," http://www.solidum.com/papers/pax-pdel/pax-pdl-00.html, Sept. 30, 1998.

[23] N. Brownleee, "SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups," Internet draft draft-ietf-rtfm-ruleset-language-07.txt, Aug. 1999

[24] L. Cholvy and F. Cuppens, "Analyzing Consistency of Security Policies," 1997 IEEE Symp. Security and Privacy, 1997.

[25] J. B. Michael *et al.*, "On the Axiomatization of Security Policy: Some Tentative Observations About Logic Representation," Database Security VI: Status and Prospects, 1992.

[26] J. Moffett and M. Sloman, "Policy Hierarchies for Distributed Systems Management," *IEEE JSAC*, vol. 11, 1993, pp. 1404–14.

[27] E. Lupu and M. Sloman, "Conflicts in Policy-based Distributed Systems Management," To appear in *IEEE Trans. Software Engineering*, Special Issue on Inconsistency, 1999.

[28] M. Sloman, "Management Issues for Distributed Services," *Proc. IEEE 2nd Int'l. Wksp. Services in Dist. and Networked Environments*, June 1995, pp. 52–59.

[29] J. Moffett and M. Sloman, "User and Mechanism Views of Distributed Systems Management," *IEE/IOP/BCS Dist. Sys. Eng. J.*, vol. 1, no. 1, Aug. 1993, pp. 37–47.

[30] M. Sloman, "Policy Specification for Programmable Networks," *First Int'l. Working Conf. Active Networks*, June 1999.

[31] J. B. Michael, "A Formal Process for Testing the Consistency of Composed Security Policies," Dept. of Info. and Software Sys. Eng., George Mason Univ., Fairfax, VA, 1993.

[32] The PAIR Project: Policy Analysis of Internet Routing, http://www.rsng.net/pair/, 1999.

[33] Route Server Next Generation Project, http://www.rsng.net/, 1999.

[34] G. Stone, "A Path-Based Network Policy Language," Dept. of Comp. Sci., Naval Postgrad. School, Monterey, CA, Sept. 2000.

Additional Reading

[1] B. Aiken *et al.*, "Terminology for Describing Middleware for Network Policy and Services," Internet draft draft-aiken-middleware-reqndef-00.txt, April 30, 1999.

[2] M. Blaze *et al.*, "The KeyNote Trust-Management System Version 2," IETF Network Working Group RFC 2704, Sept. 1999.

[3] A. Guillen, R. N. Kia, and B. Sales, "An Architecture for Virtual Circuit/QoS Routing," 1993 Int'l. Conf. Network Protocols, 1993.

[4] J. Kurose and K. Ross, *Computer Networking A Top-Down Approach Featuring the Internet*, Addison-Wesley, 2000, pp. 152–53.

[5] B. Leiner, "Policy Issues in Interconnecting Networks," IETF Network Working Group RFC 1124, Sept. 1989.

[6]H. Mahon, "Requirements for a Policy Management System," Internet draft draft-ietf-policy-req-00.txt, September 1999.

[7] C. Villamizar, C. Alaettinoglu, and D. Meyer, "Routing Policy System Replication," Internet draft draft-ietf-rps-dist-04.txt, Sept. 28, 1999.

[8] X. Xiao *et al.*, "Traffic Engineering with MPLS in the Internet," *IEEE Network*, vol. 14, no. 2, Mar./Apr. 2000, pp. 28–33.

[9] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control," Internet draft draft-ietf-rap-framework-01.txt, Nov. 1998.

Biographies

GARY N. STONE (stone@cs.nps.navy.mil) received a B.S. degree from the State University of New York at Buffalo in 1987, an M.S. degree from Johns Hopkins University in 1990, and a Ph.D. degree from the Naval Postgraduate School in 2000, all in computer science. He is currently a senior computer scientist with the Department of Defense at Fort Meade, Maryland. His interests are in traffic management and network protocols.

BERT LUNDY (b.lundy@cs.nps.navy.mil) received a Ph.D. in computer science in 1988 from Georgia Tech, with emphasis on formal specification and analysis of protocols. He continued to do research in formal protocol models, protocol testing, and areas of networks. Since 1998 he has been investigating the problems of telecommunications infrastructure, particularly in Mexico and Latin America, and has developed a plan to develop Internet and telephone infrastructure in Latin America.

GEOFFREY G. XIE (xie@cs.nps.navy.mil) received a B.S. degree in computer science from Fudan University, China, in 1986, an M.S. degree in computer science and an M.A. degree in mathematics from Bowling Green State University, Ohio, in 1988, and a Ph.D. degree in computer sciences from the University of Texas, Austin, in 1996. From 1991 to 1993 he worked full-time as a project engineer at Schlumberger Systems Center, Austin, Texas. He is currently an assistant professor with the Department of Computer Science, Naval Postgraduate School, Monterey, California, and leads an effort in developing an automated centralized network management system that supports guaranteed and differentiated quality of services and makes efficient use of network resources. His current research interests include network autoconfiguration, traffic engineering, and security.