Utilizing Semantic Tags for Policy Based Networking

Sethuram Balaji Kodeswaran, Olga Ratsimor, Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, USA
{kodeswar,oratsi2,joshi}@cs.umbc.edu

Filip Perich
Shared Spectrum Company
Vienna, VA 22182, USA
fperich@sharedspectrum.com

***Abstract*— Policy based networks provide high levels of flexibility by allowing definition of packet handling rules within a network, resource allocation strategies, network management, or access control. Commonly used policy specification mechanisms are, however, limited in their expressibility and rely mostly on packet headers that convey limited information about the semantics of the content. We propose a new model for policy based networking that utilizes the W3C Web Ontology Language tags carried in data packets that can provide detailed semantic information about the packet or stream. Using this model, a policy decision point can reason over these tags and infer the correct set of operations to invoke. Policies are expressed in the W3C Semantic Web Rule Language using a common ontology and take into consideration the content of the streams, relevant contextual information and external domain constraints. Using this framework, fine grained, highly specialized services can be offered within the network that are context-aware, easy to manage, deploy and verify for consistency.**

I. Introduction

Policy based networks employ mechanisms that allow network operators to specify at a high level, rules defining how packet flows are handled within a network, how network resources are allocated, define access control restrictions and levels of service. The policies are enforced by configuring the network devices with the requisite primitives so that the desired actions are performed on the data streams. One of the main challenges frequently faced is ensuring that network configuration settings are applied consistently throughout the network so that the correct actions are taken by the network devices; however, this is often error-prone and difficult to manage especially when there is a heterogeneity of network devices and management protocols. Additionally, policies that are commonly in use today are limited in their expressibility. Rules such as *"allow traffic from A higher priority over B"* and *"permit user A"* are easy to enforce but are limited in their expressibility. For networks to offer highly specialized services, administrators need to be able to specify more complex handling rules such as *"allow security surveillance video streams higher priority than webcasts"* (within an enterprise) or *"downsample any video to user A so as not to exceed 128 kbps"* (due to different levels of service or capabilities of the device associated with the user). For such policies, enforcement cannot be performed by packet header inspection alone as all the requisite details may not be directly accessible from the data packets as they are today.

To solve this issue, we define an alternate model to achieving policy based networks that provides fine grained services for network traffic, automates network configuration and eases network management. The model relies on two key components; namely a tagging mechanism that allows packets and/or streams to convey higher level semantic information that can be used in conjunction with the lower level information garnered from packet header inspection and a framework for specifying rules in an easy to use, formal model that can be checked for consistency. The process of converting the rules to the lower level primitives understood by the network devices is also handled by the framework allowing the network administrators to focus just on defining the administrative policies. In our model, applications encode data packets with descriptions conveying content semantics using the W3C Web Ontology Language (OWL)[9]. Ideally, the ontology used for this is provided by the network service provider. This description is encoded as a special header that is embedded into the data stream. Our motivation for using OWL (specifically, OWL-DL) is capability of the language to express formal semantics, defining class hierarchies and their relationships, associated properties, cardinality restrictions while still retaining decidability and computational completeness. Using OWL for ontology specification makes the framework generic, flexible and more scalable than using proprietary labeling schemes that raise interoperability issues.

Utilizing the framework, interim routers that handle the data packets, run a reasoning engine that can reason over the OWL description and invoke rules depending on the correct set of actions that need to be enforced. Our framework utilizes the W3C Semantic Web Rule Language (SWRL)[7] as the rule language which provides an easy to use mechanism for specifying event-condition-action rules which is the majority of rules envisioned for a typical network. Using this framework, content providers now provide metadata to the network that can then be used by the network providers to determine how best to handle a given packet or flow that best suites that content. While our framework allows the deployment of specialized handling routines into the network, a key differentiator between our approach and that of active

This work was supported in part by DARPA under contract W31P4Q-06-C-0395

networking[11], [2] with respect to packet handling is that unlike active networks, the metadata is not a contract on how the data should be handled but rather what the data is. The network provider retains complete control of how the packets are handled within the network and can fine tune policies to offer the best service for that type of content.

In this paper, we focus on a typical service differentiation use case and show how our architecture can be used to provide different levels of fine-grained traffic prioritization that are not feasible through traditional packet header inspection techniques. We have developed a network ontology to describe packets and flows and example policies to fine tune the levels of service that they are offered. We have also developed a simulation toolkit in NS2 to implement aspects of our proposed architecture allowing us to simulate various scenarios and how policies can be expressed to offer desired services.

II. Proposed Architecture

The policy based networks managed using our framework are envisioned as a multi-tier system. A typical enterprise can be viewed as a collection of multiple Autonomous Domains (AD) that may each be separately managed. Within each AD, there may be multiple sub-domains. In our architecture, policies can be classified as enterprise wide, specific to an AD or specific to a sub-domain within an AD. Policies in the system are distributed to the various ADs that are responsible for enforcement of those policies within that domain and all contained sub-domains. Each sub-domain, in turn, is responsible for enforcing policies specific to that sub-domain and so on thus providing a hierarchical policy enforcement. At the lowest level of this hierarchy is an adaptation layer that is responsible for translating the high level policies into low level protocol specific configuration routines that can be applied to the various network elements that are being managed.

Built into the architecture is a policy validation mechanism. All network management policy changes specifically, adding new policies, modifying existing policies and deleting policies are first examined for correctness and validity before being accepted into the system. Through this checking, the overall consistency of the system can be maintained. The validation actions themselves are expressed through policies set forth typically by an enterprise network administrator. Figure 1 illustrates the various components of our proposed architecture as applied to a generic enterprise comprised of several ADs managed by different administrators (this could be due to the departments being in different geographical locations, company policy etc.) and may potentially contain equipment from multiple vendors (our terminology is derived from [4]).

The *Enterprise Policy Data Store* (EPDS) is a central repository of all of polices that govern the enterprise. The EPDS contains a superset of all policies for each policy repository within the system. In addition, the EPDS stores any policies that govern the Enterprise Policy Arbitrator (EPA). Internally, the EPDS can be setup to be a hierarchical data store where for example, enterprise wide policies can be stored separate from departmental policies. Each departmental

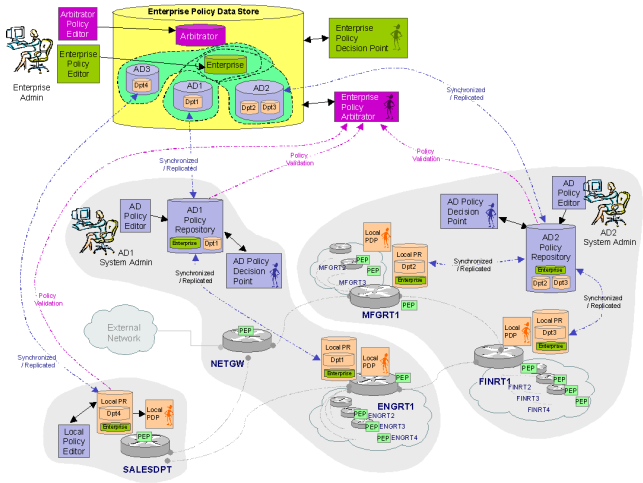

Fig. 1. Proposed Architecture

Policy Repository (PR), as part of its initialization, will contact the EPDS to obtain the set of policies that are relevant to this PR. This includes the enterprise wide policies and any department and sub-department specific policies. The EPDS is constantly synchronized with the PRs in the system. Any policies approved for addition to a PR will be also forwarded to the EPDS so that if that PR were to crash, it can come back up and retrieve its original state from the EPDS.

The *Enterprise Policy Arbitrator* (EPA) validates any new policies that are being added/removed/modified to the system. The EPA is responsible for conflict resolution, dominance check, bounds check, relation checks, consistency checks, feasibility check etc. The EPA uses the policies stored within the EPDS to perform these validation checks. The EPA is governed by a set of its own policies that define how it does the arbitration, validation and other checks. In this manner, the EPA ensures that any policy entered into the system conforms to certain global system constraints. All policies that are submitted to a PR are first forwarded to the EPA. The EPA is responsible for validating this policy against the policies in the EPDS and either admits or rejects this submission. Based on the response received from the EPA, the local PR either installs the new submission or rejects it.

The *Policy Repository* (PR) is a data store for a collection of policies. Typically this could be a AD specific PR (AD-PR), department or sub-department specific PR (referred to as local-PR) etc. The AD-PR contains all policies specific to the enterprise and any policies specific to all departments (and sub-departments) that are part of this AD. Each PR in the system is synchronized with its parent PR and at startup, retrieves all its policies from this parent. In this hierarchy, the EPDS is the root parent. An AD-PR will retrieve all policies from the EPDS including all enterprise wide policies, all AD specific policies and all policies for any contained departments or sub-departments. A departmental local-PR on startup will contact an AD-PR to retrieve all relevant policies, including enterprise specific policies, AD specific policies and

any department specific policies. Polices can be added, deleted or modified from a PR through a Policy Editor. Any such changes (regardless of whether the change is in an AD-PR or a Local-PR) are first forwarded to the EPA for validation. If they are consistent, the EPA applies these changes into the EPDS. This addition will propagate to the PR chain so that all the PRs in the hierarchy are updated. The synchronization between the EDPS and the PRs (and between AD-PRs and Local-PRs) can be through database replication or application level write throughs with distributed transaction support.

The *Policy Decision Point* (PDP) is the entity that is responsible for reasoning over the network traffic utilizing the content metadata, network state and other contextual information available to it and determining the policies that need to be enforced. Each PDP operates with policies that are stored in a corresponding PR. PDPs can be at different levels, administrative domain (AD-PDP), department or sub-department specific PDPs (local-PDP) etc. The PDP is responsible for reasoning over the policies (using its Configuration Reasoner) in its local-PR and translating them into commands that can be sent to PEPs for enforcement (to the PEP's Configuration Conformance Enforcer). Additionally, the PDP is responsible for reacting to events coming from managed PEPs or subordinate local-PDPs that cannot be resolved at the local-PDP level. In this manner, the PDP acts as the decision making entity within the framework, the decisions being made at multiple levels depending on the severity of the trigger. The PDPs closer to the device deal with policies that are low level, fine grain and possibly device/protocol specific and the PDPs higher up the tree deal with more abstract and aggregate policies.

The *Policy Enforcement Point* (PEP) is the entity responsible for enforcing the policies at the device level. It resides on the managed devices and is responsible for installing and monitoring the health and status of a network device. PEP's main responsibilities and actions are:

- To request and store its configuration from the local-PDP that is responsible for this device
- To delegate any policy decisions to the local-PDP by extracting content metadata from data packets and adding to this description, any additional information that may be useful to the local-PDP
- Report errors and status updates to the local-PDP

The PEP is a vendor/device specific network management protocol. The PEPs collectively form the adaptation layer to abstract any device specific details into a normalized interface represented using the standard system ontology.

The *Network Ontology* (NetOnto)is the OWL ontology specified by the service provider or enterprise that is used to mark up the content of data packets conveying information such as application profiles (security requirements, delay, jitter etc.) and user profiles (customer paying more for service, end device capabilities). By using OWL rather than simple XML, the language is semantically richer and highly extensible which is very important especially when we have interdomain interactions (such as peering arrangements, SLAs etc). Policies are written using the concepts defined in NetOnto using SWRL as the rule language. The content descriptions are carried in the packet headers either as directly embedded, contain a URL to the content description or use UUIDs that imply a certain type of content. A PEP extracts this description and adds to it, any extra contextual information including aspects such as network state (congestion, link failures etc), network technology (wired, hybrid, MANET, cellular) etc. This information is then sent to the PDP and actions are invoked based on the response. The response back from the PDP will cause specific configuration to be installed by the PEP on the device (for example, updates to IPtables, addition of static routes and priorities, setup a label switched path etc).

The *Policy Editor* (PE) is the component that is used by a system administrator to view, add, modify and delete existing policies. The interface is typically a GUI allowing for ease of operation. PEs can be at different levels. The Enterprise Policy Editor (E-PE) provides a way for an enterprise system administrator to specify enterprise wide policies. An Arbitrator Policy Editor (A-PE) allows an enterprise system administrator to specify policies that drive the EPA. Similarly, an AD Policy Editor (AD-PE) allows an ADs system administrator to specify AD specific policies. In general, a user uses the PE to request changes to be made to a PR. This request will pass through the EPA validation and the user receives an acknowledgement of whether or not the requested change is allowed or rejected. Additionally, the PE also offers views of the managed network such as topology views, status of network devices and links etc.

Utilizing this framework, devices can now be deployed in a network and policies specified that can provide specialized services (content adaptation, forwarding priorities, resource reservations etc) to data packets. At each interim device, packets are inspected to see if they carry a semantic header. These packets are then offloaded to a separate forwarding path. The semantic header is extracted, any additional contextual detail available to the PEP is added to complete the OWL description and sent to the PDP for reasoning. We use OWL's abbreviated XML encoding format for this purpose. The PDP runs a reasoner that takes the OWL description and SWRL rules to determine the actions that need to be initiated. This information is then conveyed back to the PEP to be applied to the offloaded packets (and possibly to the express lane) to realize the necessary policies.

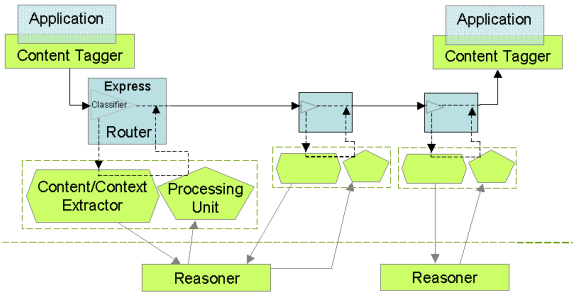

Fig. 2. Packet Flow

III. Typical Use Case

The use case we consider in this paper is that of a secure enterprise that wants to enforce prioritization on types of content that can flow across the links comprising its network. The enterprise has profiled its network and assessed security credentials to all the links and routers. As an example, a link that is fully within the premises of the enterprise (physically secure) is assessed as a "safe" link, one that is a VPN running over an external service providers network may be assessed as "potentially unsafe" while a wireless RF hop may be assessed as "unsafe". The enterprise applications are "smart" and can encode content level tags into the data packets that carry semantic information about the content as well as the application/user/device. For this example, as we are interested in the security semantics, applications additionally provide information about the sensitivity of the content (such as secret, top secret or normal), type of content and the security credentials of the context within which they run. For such an enterprise, the following policies would be appropriate:

- Only "Safe" links can be used to carry "TopSecret" data
- All data over "Open" links need to be encrypted
- Restrict multimedia flows in the network to max of 75% link capacity
- Allow admin traffic preferential service over network backups
- Allow user access to data only if user clearance is high enough

IV. Simulation Toolkit

We used NS2 to simulate such an enterprise. The network topology considered was a random network with links classified with a "security" tag that defined their safety levels. We assume the nodes belong to a single AD and run a link state routing protocol. We modified the standard FTP/CBR applications to allow for the specification of semantic descriptions into the packet streams. For the network ontology, we used Protege as the editor for specifying our ontology. Jess was used as the reasoning engine. The choice of Jess was mainly motivated by its easy integration with Protege. Other reasoning engines can be used as a replacement if desired.

To begin, we defined an ontology to represent our enterprise. The ontology is available online at [1]. In our implementation, our ontology also contains special instances of classes representing the various actions that a PEP should take such as dropping data, encrypting data etc. These special instances also contain the low level primitive commands that need to be invoked to realize the necessary behavior. In our case, these commands are expressed as a snippet of Tcl code that can be evaluated by NS2. For example, a policy such as *All unencrypted secret data over "open" links need to be encrypted* can be expressed logically in SWRL as:

```
DataTraffic(?d) ∧
datasensitivity(?d,?sensitivity) ∧
Secret(?sensitivity) ∧
encryptionstatus(?d,?encryptstatus) ∧
UnEncrypted(?encryptstatus) ∧
nextHop(?d,?nexthop) ∧
securityLevel(?nexthop,?securitylevel) ∧
Open(?securitylevel) ∧
EncryptData(?action)
  → inferredAction(?d,?action)
```

The *EncryptData* instance has the following Tcl command encoded in it indicating the device understandable actions that need to be taken.

```
set clsfr [ get-classifier $interimRouterId ]
$ns at [$ns now] ``$clsfr install-interceptor
      encryptdata $flowid $srcId $sport
      $destId $dport $qdelay $overhead''
```

Using this methodology, we can now define the various actions that a PEP could take and assign to each of these actions, the corresponding primitive commands (Tcl snippets). The PDP was implemented as a Java process that received OWL streams from a client PEP (a network router within NS2), invoke the reasoner and send back the Tcl commands depending on the actions that needed to be invoked. The PEP (NS2) would then excute the commands received from the PDP. The Protege IDE served the role of a Policy Editor. Using this framework, we implemented our typical use case scenarios.

A Traffic differentiation use case. In this experiement, a typical dumbell shaped topology was chosen with senders on one side and receivers on the other. The routers were configured to support class based queues. We simulated VBR and CBR streams over the links using applications that could embed tags into the data packets. Our semantic flow classifier, upon receiving a packet, constructs an OWL description of the stream. This information includes information gathered from the packet itself and additional contextual information such as current queue lengths, next hop and network topology information. The reasoner then reasons over the streams to identify the correct classes that need to be applied to the packet. Figure 4 and Figure 5 show the observed goodput of a mixture of 2VBR and 2CBR streams. In Figure 4, the network is responding by inferring that there is a high priority admin traffic (surveillance video) that needs to be given the available bandwidth currently being used by other background admin traffic (network backups in this case). In Figure 5, the network recognizes that the CBR streams are user video streams and limits the impact of this traffic to network backups allowing the network backups to proceed as before. Using semantic tags and associated rules, the reasoner is smart enough to differentiate between a network surveillance video (admin traffic) and a regular movie clip which traditional techniques such as packet header inspection cannot because both will appear as standard MPEG flows. In addition, the reasoner can also specify intra-flow packet prioritization driven by policies [8].

```
AdminTraffic(?d) ∧
CodeData(?action) →
   inferredAction(?d,?action) ∧
   newflowid(?action, 2)
```

```
UserMultimediaTraffic(?d) ∧
CodeData(?action) →
    inferredAction(?d,?action) ∧
    newflowid(?action, 12)
```

```
- <rdf:RDF xml:base="http://www.cs.umbc.edu
  - <owl:Ontology rdf:about="">
      <owl:imports rdf:resource="http://www.
      <owl:imports rdf:resource="http://www.
    </owl:Ontology>
  - <umbc:VBR rdf:ID="VBR_1">
      <umbc:contentType rdf:resource="http
      <umbc:encryptionstatus rdf:resource=
      <umbc:datasensitivity rdf:resource="ht
      <umbc:nextHop rdf:resource="http://ww
      <umbc:sourceRoute rdf:resource="http
      <umbc:interimRouter rdf:resource="ht
      <umbc:srcPort rdf:datatype="http://ww
      <umbc:src rdf:resource="http://www.cs.
      <umbc:destPort rdf:datatype="http://ww
      <umbc:dest rdf:resource="http://www.c
      <umbc:transport rdf:resource="http://w
      <umbc:fid rdf:datatype="http://www.w3
    </umbc:VBR>
  </rdf:RDF>
```

Fig. 3. Sample OWL Description

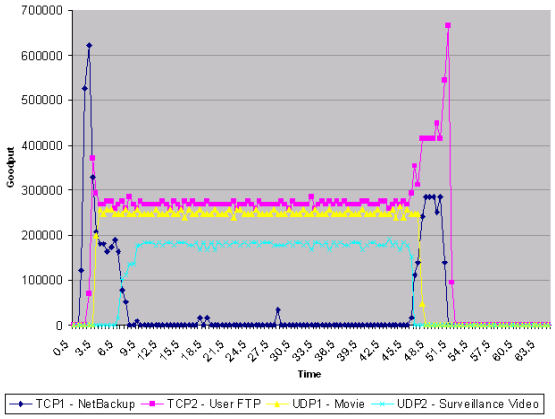

Fig. 4. Observed Goodput

V. Related Work

Policy based networks and approaches have been the focus of extensive research in recent years. Quality of service oriented initiatives such as Intserv[6] and Diffserv[5] rely on policies to drive flow classification, admission control, resource reservations etc. However, the policies used are limited in their expressibility and restricted to traffic forwarding semantics with little support for features such as content adaptation, specialized routing etc. In this respect, the Active Networks[11], [2] took the approach of allowing a more generic per packet handling semantic with the packets determining what the router should do with them. A key differentiator between our approach and the traditional AN approaches is that while we rely on tags in the packet stream, it is the router (using its specified policies) that controls how the packet is handled and not the other way. [10], [3] are relevant work directed at development of formal ontologies and tools to facilitate network management and control.

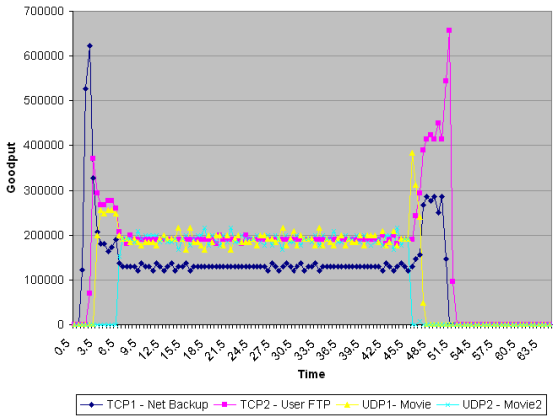

Fig. 5. Observed Goodput

VI. Conclusion

In this paper, we present our architecture for realizing intelligent policy based networks. The framework relies on the use of smart decision points that can reason over contextual descriptions of data streams to invoke policy decisions that are most appropriate for that stream. Packets are augmented to carry semantic rich tags decribing their content. A simulation framework and an example ontology is also presented with its application to an use case example. We are currently implementing additional content adaptation capabilities that can be tested out in the simulator and looking into mechanisms for policy aggregation, conflict resolution, validation and prioritization.

References

[1] http://www.cs.umbc.edu/ kodeswar/ontologies/NetworkOnto.owl.

[2] D. S. Alexander, W. A. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare active network architecture. *IEEE Network Magazine*, 12(3):29–36, 1998. Special issue on Active and Controllable Networks.

[3] F. Alonso, R. Fernandex, S. Frutos, and J. Soriano. Defining a semantic web-based framework for enabling automatic reasoning on cim-based management platforms. *International Journal of Computer Science*, 1(2):99–110, 2006.

[4] A.Westerinen, J.Schnizlein, J.Strassner, M.Scherling, B.Quinn, S.Herzog, A.Huynh, M.Carlson, J.Perry, and S.Waldbusser. RFC 3178: Terminology for Policy-Based Management, November 2001.

[5] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, December 1998. Status: INFORMATIONAL.

[6] R. Braden, D. Clark, and S. Shenker. RFC 1633: Integrated Services in the Internet Architecture: an Overview, June 1994.

[7] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. Technical report, W3C Member submission 21 may 2004, 2004.

[8] S. B. Kodeswaran and A. Joshi. Content and context aware networking using semantic tagging. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, page 77, Washington, DC, USA, 2006. IEEE Computer Society.

[9] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview. Technical report, W3C Recommendation 10 February 2004, 2004.

[10] S. Quirolgico, P. Assis, A. Westerinen, M. Baskey, and E. Stokes. *Toward a Formal Common Information Model Ontology*, volume 3307, pages 11–21. 2004.

[11] D. Wetherall, J. Guttag, and D. Tennenhouse. Ants: A toolkit for building and dynamically deploying network protocols, 1998.