2009

Implementation of Network Services Supporting Multi-Party Policies

Santosh C. Proddatoori
University of Massachusetts Amherst

IMPLEMENTATION OF NETWORK SERVICES SUPPORTING MULTI-PARTY POLICIES

A Thesis Presented

by

SANTOSH CHANDRA PRODDATOORI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

September 2009

Electrical and Computer Engineering

IMPLEMENTATION OF NETWORK SERVICES SUPPORTING MULTI-PARTY POLICIES

A Thesis Presented

by

SANTOSH CHANDRA PRODDATOORI

Approved as to style and content by:

Tilman Wolf, Chair

Weibo Gong, Member

Ramgopal Mettu, Member

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to take this opportunity to thank Prof. Tilman Wolf, who has been nothing less than a perfect guide during the last two years of my Research work. I would also like to thank him for providing me with an opportunity to work on such an interesting project.I thank all my friends and colleagues in the Network Systems Lab - Xin Huang, Qiang Wu, Shashank Shanbhag and Anindya Misra - for their time and effort to help me out during various stages of my project. Finally, I would also like to thank my family and my friends for their moral support and encouragement in times of need.

ABSTRACT

IMPLEMENTATION OF NETWORK SERVICES SUPPORTING MULTI-PARTY POLICIES

SEPTEMBER 2009

SANTOSH CHANDRA PRODDATOORI

B.E E.C.E, OSMANIA UNIVERSITY, INDIA

M.S.E.C.E, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Tilman Wolf

Next-generation network architectures support complex services in the data-path of routers. A key challenge is the integration of multiple policy constraints from senders, receivers, and network providers when using such services. We introduce a multi-party service specification framework based on our service socket API. We illustrate the operation of this approach in an IPTV scenario that uses a video transcoding service implemented on a Cisco ISR platform.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS ..... **iii**

ABSTRACT ..... **iv**

LIST OF FIGURES ..... **vii**

CHAPTER

1. INTRODUCTION **1**

1.1 Motivation 1

2. RELATED WORK **4**

3. NEED FOR NETWORK SERVICES **6**

4. NETWORK SERVICE ARCHITECTURE **9**

4.1 Network Services Concepts 9
4.2 Design of Network Service Architecture 13

4.2.1 Service Controller 13
4.2.2 Service Node 16

4.3 Service Sockets 17

5. NETWORK POLICIES AND POLICY-BASED NETWORKING **18**

5.1 What are Network Policies? 18
5.2 Multi-Party Service Specification using Data semantics 20

5.2.1 Data Semantics Specification Syntax 22
5.2.2 Network Service Policy Management 24

6. PROTOTYPE IMPLEMENTATION AND RESULTS **33**

6.1 Why Cisco ISR? 33
6.2 Prototype Setup 37

6.2.1 Introducing Transcoding Services based on Policy Rules 37

6.3 Results 45

7. CONCLUSION AND SUMMARY **50**

7.1 Contribution 50
7.2 Future Work 51

BIBLIOGRAPHY **52**

LIST OF FIGURES

Figure **Page**

4.1 [20] Network with Data Path Services. Connection requests specify sequence of services that need to be allocated by Network. 12

4.2 [13] Network Service Architecture. 12

4.3 [13] Network Service Router System. Service Controller functionality is shown in the upper half, Service Node functionality is shown in the lower half. 14

5.1 Extension to Service Socket API that processes and appends a semantic Header to the request packet. 25

6.1 Service Composition Process. 35

6.2 Prototype Setup of NSA. 37

6.3 Results showing the introduction of transcoder services in the data path. 39

6.4 OWL description of Service specification. 42

6.5 Policy rules defined in SWRL. 44

6.6 Space-time description of the service specification example with and without transcoding service enabled. 46

6.7 Prototype Implementation of NSA on Cisco ISR with and without transcoding service enabled. 47

6.8 Data-rates when a video of resolution 1280x544 is transcoded to 320x240, 640x480 and then no transcoding 49

CHAPTER 1

INTRODUCTION

1.1 Motivation

The current Internet architecture has been the best example of a computer network. It is still dominant because of its ability and flexibility to add and support new protocols, technologies and applications due to the increasing demand. This simple architecture provides a basic end-to-end communication service with the end-systems providing more complex functionality and the high speed routers between providing simple functionality of moving packets from one node to another. It is just a store-forward platform for data. However, this design pattern has a lot of shortcomings on the ability to adapt to the requirements of the current and future technologies. This can be best shown in the current Internet, where the introduction of new networking concepts and increase in the diversity of end systems has resulted in new applications like peer-to-peer networks, content distribution and caching architectures that go beyond the traditional end-to-end argument. These developments and applications placed within the network make it necessary for the routers to perform more complex functions.

To address these issues, these architectures need to be replaced by some new network architectures. Hence, a great amount of research is going on in the area of new network architectures that can provide the foundations of the next-generation Internet. These next generation network architectures do not need to be backward compatible with the existing Internet and hence can be considered a clean-slate design [11].

As a part of this, a new network architecture has been proposed [23] and [20] based on the concept of a network service. These network services are just like a processing function performed in the data path of the network that provide all functions needed for data communication and those placed on end-systems (e.g. TCP, UDP) along with the functions that are placed on routers (e.g. NAT). This approach makes the deployment of network services simpler and hence to implement these network services programmable routers that are capable of high-performance packet processing can be used.

The major challenge here would be the need for abstractions, which allow end system applications to make use of the features provided by the network services. This needs to be taken care without forcing any considerable complexity and allowing for transparency in the service specification and application. Another challenge here would be the composition of services and its placement within the composed set of services. When network services are composed as a sequence of operations, it poses constraints on the application of network policies on traffic. This can be best explained by the following example: When IDS service is applied as a network policy to network data, it invariably fails if the packet payloads are encrypted. To avoid these, it is very much necessary if semantics of data can be considered to decide whether to enforce a policy or work around it. As well, the semantics of data and policy requirements can be used to add new services to the earlier service composition to ensure that the policy is enforced if it is a 'strict' policy. Generally, the network policies are specified using simple rules that depend on specific values in the packet headers. But, this will not be sufficient in next generation networks, where additional services need to be added to or removed from the service compositions based on the semantics of data being sent, which cannot be extracted from the packet headers alone. This thesis work is thus based on the ways to overcome the above said challenges and support network policies by extending the service specification framework

and service sockets API [20] to include semantics of data the end-systems intend to send.

CHAPTER 2

RELATED WORK

There have been a wide number of different approaches proposed to extend and adapt the functionality of the original Internet architecture. Some of the examples are network address translation to allow IP address reuse [10], firewalls to improve security [17]. Furthermore, for providing data path flexibility (e.g., active networks [22], configurable protocol stacks [7], protocol heaps [8]) protocol extensions have been proposed. Routers have been designed with advanced features for handling packet processing (extensible workstations routers [14], programmable routers [19], and virtualized router platforms [4]). In this work, the network services describe the advanced networking functions.

An important challenge with providing new network capabilities is finding a way of allowing management and control. An interesting attempt in this case was active networks [22], which provided a powerful and very general approach to customizing packet processing. But, programming network features is a difficult task. Active networks were one of them that were difficult to control and hence the application programmers found it impossible to use. To handle the complexity of new features in the network, an attempt was made by an IETF working group to define Open Pluggable Edge Services (OPES) [6]. The end systems in such an architecture will have the option of specifying a set of data flow operations that are implemented on nodes through out the network. Here, we use the concept of pipeline abstractions [16] to provide a general method of specifying services. Another general approach to man-

aging network functionality is the SILO architecture [18], which provides mechanisms for composing custom protocol stacks [5].

This thesis work allows a more general sequential composition of services rather than just limiting to a layered composition of function. An important concept here is to use a tagging mechanism to convey semantic information from the packet data. Multiparty service composition and service placement according to network policies is the main theme of this thesis work.

CHAPTER 3

NEED FOR NETWORK SERVICES

The current Internet design has no provision for supporting dynamic deployment of new protocols. Because of this drawback, it took time for implementing several additional protocols since the initial Internet design did not consider them.

Some of the applications, which cannot be implemented on the end systems only and are characterized by the need of support by the network are listed below:

- Intrusion Detection Systems (e.g., snort [21]) is used to check several types of malicious behaviors that can compromise the network security. This includes host based attacks such as privilege escalation, unauthorized logins, network attacks against vulnerable services, data driven attacks on applications and malware (viruses, trojan horses, and worms). This mechanism dynamically blocks the traffic that is considered malicious by checking packet headers and content against signatures of well-known attacks.

- Web Switching [9] is a mechanism for distributing a single web server over several physical machines while presenting a single front-end to the outside. Here, the Http requests are parsed in packets and an appropriate server will be determined to which the request will be forwarded. As the Http request is sent only after establishing a TCP connection, the web switch also has to splice the TCP connection between client and back-end server.

- Random Early Detection (RED [12]), is an algorithm used for queue management and congestion avoidance. It basically monitors the average queue size

and then drops packets based on statistical properties. The incoming packets are accepted only when the buffer is empty. With the increase in size of the buffer, the probability of dropping an incoming packet also increases. The probability reaches 1 when the buffer is full meaning all the incoming packets will be dropped. This trend is implemented by many current routers, but only a few are used in practice.

- Network Address Translators (NAT [10]) is a technique for IP address reusing. When the network traffic flows through a router, the source and/or destination IP addresses and the TCP/UDP port numbers of IP packets will be rewritten. This is basically done to expand the set of possible IP addresses within a given network. Thus, NAT allows the usage of a single globally unique IP address among multiple hosts in a subnet. The NAT modifies the IP packets passing between the subnet and the Internet. Due to this, the number of IP addresses used by a subnet reduces and thereby extends the time before the IP address space is exhausted.

- Firewalls [17] is a network security component incorporated in most networks for inspecting the network traffic flowing through it and deny or permit the passage based on a set of rules. As a result, network traffic that compromises the security of hosts on the network (such as port scanning) can be blocked. Firewall rules can be numerous and complex, which requires significant computational power on the firewall to keep up with typical access link speeds.

The above mentioned systems provide an example of how new protocols slowly forced their way into the network. This shows an expansion to the traditional store-process-forward networking. The upcoming issues of security, resource management and isolation, and programming complexity limit the practicality of such a general approach to network services. A more constraint model of programmable routers is

a more realistic scenario for deployment. In programmable routers, a selected set of features are installed by an administrator and end-systems may choose if their communication utilizes these services.

The advantage of this approach of explicit service features in the network is that such services can be visible to the end-system and thus managed accordingly. The management issue of middle boxes has been addressed by the IETF Open Pluggable Edge Services (OPES) working group [15] and the IRTF End-Middle-End (EME) working group.

CHAPTER 4

NETWORK SERVICE ARCHITECTURE

The concept of network services and network service architecture provides the necessary context for our multiple service policy design. The design of network service architecture is a prior work [13] and the same concept is used in this work for multiple service policy design by modifying the service sockets API and the service specification syntax to include data semantics aware multiparty service composition.

4.1 Network Services Concepts

Network services form the basic components of the functions performed on a connection. The data is first encoded and then the encoded data is transferred in various ways. The end system applications in a service-centric end-to-end architecture specify the information transfer that is desired and the network (in assistance with the end systems) determines how the data needs to be handled. There are three major aspects that need to be considered in this process:

- Data Semantics and Encoding: The information that needs to be transmitted has to be encoded into data. This is already given in many of the cases, such as in a file transfer.

- Data Transmission: The transfer of data between end-systems and network systems corresponds to the traditional functionality of a network.

- Data Processing: Data needs to be processed on its way to the destination to support advanced services. This includes tasks like modification of the transferred data.

As mentioned in the last point above, the existing Internet architecture differs in having data processing as its key networking function. The network service architecture thus permits the idea of processing through out the network rather then just limiting to the end systems as is done in our existing Internet architecture. The desired functionalities and performance properties will be then handled by the network services placed in the data path of the network.

The encoding of information will allow the system to maintain semantic information on the bits that are being transmitted. The data is then modified by the network to transfer the information in an efficient manner. If we consider the example of web caching, a proxy could intercept web requests and respond with a local copy of a document. This results in transferring the same information without actually involving an end-to-end data transfer. As this could be done transparently by the network, there is no requirement of any configuration on the end system.

A network service can receive, store, process and transmit data that is sent over the network. Services leverage semantic information about a data stream to implement different functionalities. Some of the examples of network services are:

- Reliability: This includes lossless transmission of data by using the mechanisms like buffering and acknowledgment-based retransmission of lost data as is done in TCP.

- Privacy: This includes encryption and decryption services between end-systems (similar to SSL) or subnets (the same as VPN).

- Congestion Control: Controlling the rate at which the data is being sent according to the state of the sub-network to avoid congestion that leads to the loss of data.

- Security: This includes various mechanisms to detect and mitigate attacks. Intrusion detection systems, firewalling and payload scanning are some of these mechanisms.

- Quality of Service: This includes prioritized forwarding of data based on service requirements.

- Multicast: This includes duplicating the data and transmitting it across multiple links.

- Payload Transcoding: This service can adapt the content that is transferred depending on the semantics of the transferred information. A simple example of this kind is a video transcoding application, which can reduce the resolution of a large image so that it can be down sampled for display on a cell phone. This is implemented here as one of the services on a Cisco ISR.

A significant point to note here is that these services are implemented in the data path of the network and act on all (or a subset of) packets. As the networking functions are decomposed into basic services, connections can then compose a custom sequence of such services to be applied to the packets transferred in this connection. The end-system applications will make an efficient use of functionality of the network from such custom compositions.

The figure 4.1 illustrates an example of a network service connection. A connection setup request with two services is shown at the bottom of the figure. During connection setup, if such a request is provided by the end-system to the network, the network then identifies the nodes that are capable of performing the requested

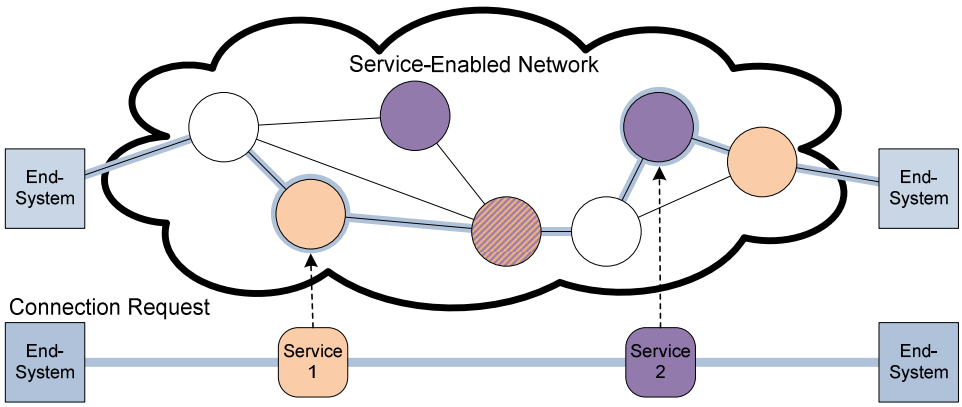

Figure 4.1. [20] Network with Data Path Services. Connection requests specify sequence of services that need to be allocated by Network.

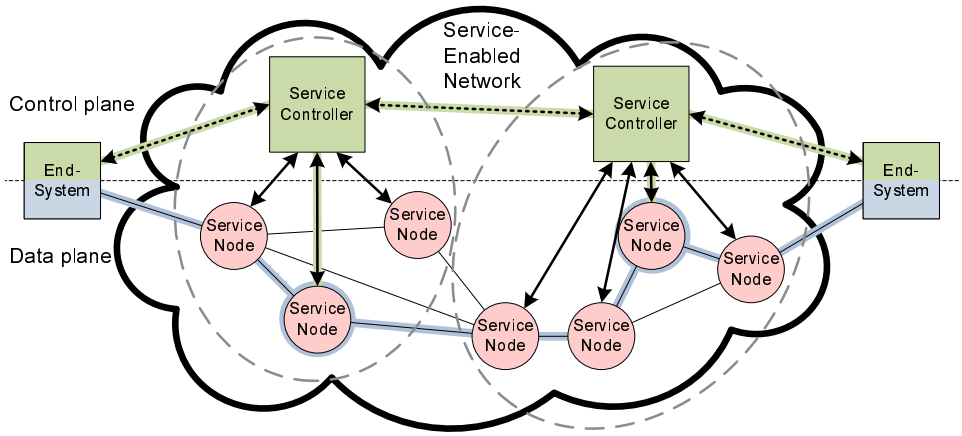

Figure 4.2. [13] Network Service Architecture.

services (denoted by corresponding colors) and chooses one for each service (arrows). The connection is then established between the end systems such that the traffic traverses these nodes on the way from one end-system to the other. The service nodes are informed about the kind of processing the packets of a particular connection require during setup. When the packets are sent via this connection, the correct set of services is performed along the way.

4.2 Design of Network Service Architecture

The high level design of the service-centric network is illustrated in figure 4.2. The control plane is shown on the top of the figure and the data plane is shown on the bottom. To set up a service-based communication, an end-system initiates the connection by sending a service specification to its local Service Controller using the service socket API. The service controller manages several service nodes. This controller allocates processing services to service nodes that it manages and arranges the data transfers between them. The service specification describes the functional requirements of a connection and lists services that are to be applied in order, to the data that the end-system intends to send. For services that cannot be allocated to nodes that the controller manages (either due to resource limitations or due to placement constraints specified by the end-system), the request is forwarded to a neighboring downstream service controller that is along the path to the destination. The service controller identifies the service nodes that are capable of performing the requested services and informs them about the kind of processing that needs to be applied to the packets belonging to a particular connection. When the path is set up, the end-system is notified by the local service controller about the connection setup and the next hop node it should send the data packets to. Therefore, the end-system can then send the data packets through this hop-by-hop connection and the correct set of services are applied to the data packets along the connection.

The section below describes in detail the in-built functionalities of a service controller and a service node, which form the key components of the network [13]. Figure 4.3 illustrates the details of these components.

4.2.1 Service Controller

The service controller consists of four major components: resource management (tracking of available processing and memory on service nodes), connection man-

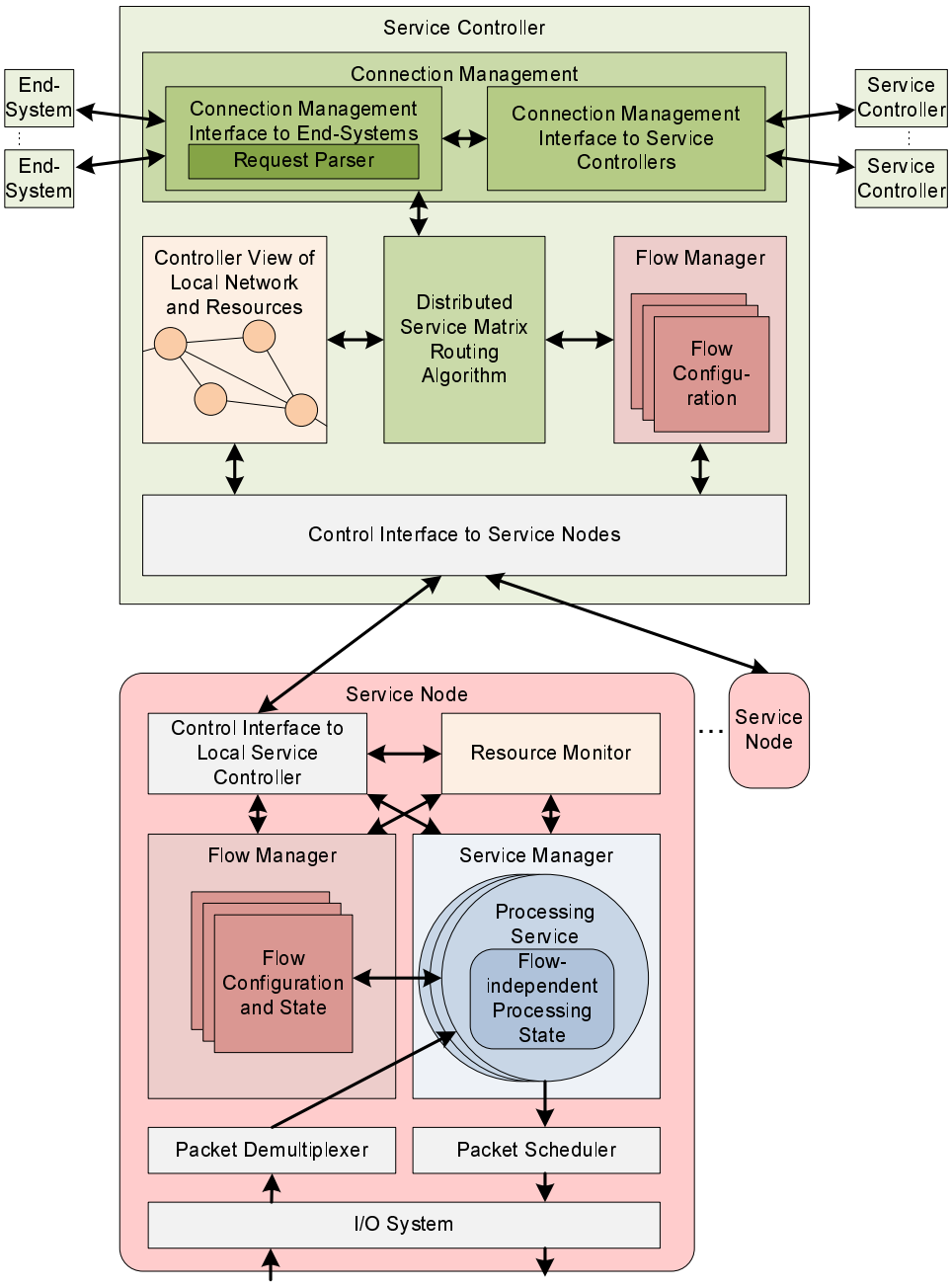

Figure 4.3. [13] Network Service Router System. Service Controller functionality is shown in the upper half, Service Node functionality is shown in the lower half.

agement interfaces (between end-systems and other service controllers), flow specific components and control interface to service nodes.

Connection Management Interfaces communicates with end-systems to receive requests for service-based communications. An active pipe abstraction is used as the specification notation that provides a mechanism for expressing services, their parameters and placement constraints in a textual string ([16, 23]). During the connection setup, this string is communicated to the service controller. The string is then analyzed by the request parser component and translates this into a representation suitable for the mapping algorithm. The advantage of the active pipe notation is that services that are not handled by the service controller can be passed on to another service controller downstream.

Resource management component tracks the available resources on local service nodes. This information is needed to decide if any new service requests can be accepted. This can be implemented in two ways: open-loop and closed-loop. In open-loop, available resources are estimated based on previous allocation, nodes do not provide direct feedback on their actual state that leads to many problems, especially when resource requirements vary depending on changes in flow bandwidth, packet payloads, etc. On the other hand in a closed-loop implementation, nodes periodically report about their available resources, memory and hence this method is preferred.

Flow Setup and Management consists of a mapping algorithm that is used for determining service placement and the flow manager. The service request that is translated by the request parser is used by the mapping algorithm and attempts to map it to local service nodes.

Service Node Interface communicates with the service nodes that are managed by a service controller. It manages the connection setup by sending control messages and receives updates on available resources.

4.2.2 Service Node

A service node is composed of five major components: flow manager, control interface to the service controller, service manager, resource monitor and data I/O system.

Service Controller Interface communicates with the service controller to receive connection requests. It then sends updates on available resources. It acts as the complement component of service node interface on the service controller.

Flow manager takes care of setting up appropriate flow classification rules in the packet demultiplexer to ensure that each flow receives the appropriate service processing. It also keeps track of which flows are currently using service on the service node. The flow manager maintains the state between packets across the flow for the services.

Service Manager takes care of managing the implementation of network services. For this, processing resources are made available for all packets that are passed from the demultiplexer. Service manager also manages the flow-independent per-service state.

Resource Monitor monitors the operation of the service manager and reports the available resources to the service controller. It takes care of resetting services and free up processing resources and memory in case of the resources locked up due to erroneous behavior. This may impact the per-flow state and hence needs to be coordinated with the service controller.

The Data I/O system takes care of receiving and transmitting the actual packets that are forwarded and processed by the service node. The access to the outgoing link can be controlled by the system through the packet scheduler.

4.3 Service Sockets

Service sockets form the main interface between the applications and the network service architecture. These sockets are conceptually similar to our traditional TCP and UDP sockets, but provide a mechanism for exchanging a service specification. The service specification informs the network what services are required by the application.

CHAPTER 5

NETWORK POLICIES AND POLICY-BASED NETWORKING

5.1 What are Network Policies?

A network policy is a guideline or a rule describing how the network ought to resolve the resource conflicts that are a natural consequence of the interactions between the users and different applications available on the network. Policies can dictate, which network resources and applications are to be accessible to which users, as well as classify different applications and users into multiple categories and give preferential treatment to some users/applications over others. As an example, a network policy may state that transaction-oriented business applications should be considered more important than random web-surfing. Another network policy may state that communication across two machines be encrypted using IPSEC.

Most of the network policies are enforced on a network with the simple idea of keeping the bad guys out, i.e. for maintaining security. They are meant to govern data access, web-browsing habits, use of passwords and encryption, email attachments and many more.

With the convergence of data, telephone and video traffic in the same network, it is a challenge to manage traffic so that one kind of service doesn't preempt another kind. Using policy statements, network administrators can specify which kinds of service to give priority at what times of the day on what parts of their Internet Protocol (IP) based network. This kind of management is often known as Quality of Service.

A Policy-based network is formed by policies that basically consist of two components:

- A set of conditions under which the policy applies. This might include parameters such as user name, addresses, protocols and applications types.
- A set of actions that apply as a consequence of satisfying (or not satisfying) the conditions including bandwidth guarantees, access control, service load-balancing, cache redirection and intelligent routing.

These conditions and actions consist of a series of passive and active components on the network. In addition, this would include a policy manager that is the central policy administration and directory repository point and a policy enforcer that consists of remote active management components that make up the local policy decision and enforcement points throughout the local wide-area networks.

The Policy Manager takes care of the policy administration. It consists of a directory database where all the policy information will be available. The policies stored in the directory database are then translated to network actions and policies. The Policy Manager will also identify all policy manageable enforcement devices in its domain. It responds to requests from policy enforcement devices for specific policy information. In the case of nodes that have local decision capabilities, it will act as a higher level administrator of policies. It provides added capabilities for coordinating discovery and management of specific policy enforcement devices. It tracks changes to the directory database and relays the information to the policy enforcers in its domain.

The Policy Enforcer can be a simple router that makes policy decisions based on a field in a particular tagged packet. Alternatively, the Policy Enforcer may be a piece of equipment that locally consolidates and analyzes traffic flows and network conditions to perform complex network actions such as:

- Traffic Conditioning and Shaping: This includes things such as traffic prioritization, traffic guarantees and bandwidth management.
- Policing: This includes access control, user authentication and remote login.
- Tagging/Signal Provisioning: This includes translating and relaying signal-provisioning information (RSVP, Differentiated Services, 802.1P, MPLS) through the network.
- Server Resource Control: This includes advanced enforcement capabilities such as server load-balancing and cache redirection.

In this work, the service controller (as shown in the figure 4.2) plays the role of a policy manager and then the service controller identifies one of its nodes that runs the equivalent service as its policy enforcer.

5.2 Multi-Party Service Specification using Data semantics

The service specification of the network service architecture [20], expresses services as a pipeline. It consists of a source and a sink that form the end systems. As a simple example, an end-system application that wants to send data to 192.168.1.2 on port 80 and as well wants to have compression/decompression (with Lempel-Ziv algorithm) service pairs to be applied to its data packets along the way can use the following service specification:

```
*:*>>compression(LZ)>>decompression(LZ)>>192.168.1.2:80
```

The services have to be applied in the order mentioned in the service specification and the intermediate services can appear any number of times (again following the order as mentioned in the service specification). However, the service specification does not place any constraints on where the above mentioned services should be placed in the network, i.e. in the service specification example above, the placement of compression

and decompression services is at the discretion of the network and is based on the availability of the service and resources in its local network. When the service is not available in the local network, the service specification can be forwarded to a neighboring service controller where the service is known to be available.

But there can be several constraints that are placed on the application of network service policies on traffic due to the composition of network services as an atomic sequence of operations. The following example will explain the above scenario: consider a service specification used by an end- system application that wants to use the encryption/decryption service pair.

```
*:*>>encryption(AES)>>decryption(AES)>>192.168.1.2:80
```

From the above service specification , it clearly shows that any network policy that enforces the incorporation of an intrusion detection service on the traffic invariably fails if the packet payloads are encrypted. This can be avoided by enforcing a network policy before the encryption service is applied to the data traffic. But, again a network policy such as an IDS service policy cannot possibly be applied if the end-system itself sends encrypted data traffic. In general, simple rules are being used for specifying current network policies in network. These rules are based on specific values in the packet headers. A simple example in this case will be that of a QoS Priority based forwarding policies where the decisions are made based on Source IP. Next-generation networks have a more challenging functionality and hence the above mentioned simple rule based network policies are not sufficient as the focus is also on service policies rather than solely on forwarding and services need to be added or removed from service compositions. A relevant solution for this will be to include semantics of the data while making service policy decisions. For example, an end-system sending a request to the service controller needs to incorporate the semantics of the data it intends to send in the service specification. The service controller can then make informed decisions on whether to enforce a policy (in case of 'strict' policies) or to work around

it depending on the semantics and service policies from the service specification sent by the end-system. Hence, in order to support these policies, we extend our service specification framework and service sockets API [20] to include semantics of the data sent by the end-systems.

5.2.1 Data Semantics Specification Syntax

Here, we have considered the service specification syntax from [20], which we have modified through this work to incorporate data semantics. It consists of the following entities:

- Source/sink: These are represented by a sequence of IP addresses and port numbers separated by a ':'. The source may or may not have the IP address and/or port number specified. For example, both 192.168.1.2:80 or *.* are valid.

- Network services: The services that needs to be performed by the network service architecture has to be specified along with its configuration parameters with the name of the service followed by a sequence of parameters in parantheses. For example, the syntax: encryption(AES:key=128), denotes that encryption is a service, and 128-bit AES is the algorithm used for encryption (configuration parameters). In the same way, multiple parameters can be specified separated by a comma.

- Director: This indicates the symbol '>>' that is used to specify the sequence in which the services need to be applied to the data.

In this work we have also considered the data semantics that have been incorporated in the specification. Thus, the end-system that requests services from the network appends to the specification the semantics of the data (such as the kind of data, data attributes) that it intends to send to enable the network to make informed policy decisions based on both data semantics and header values extracted from the

request packet. The data semantics are specified as an extendable list of semantic indicators that are used to represent the data. It is vital that the list is extendable because this makes the semantics specification framework flexible and extensible with regards to new protocols and data types. This can be explained using the following simple example: An end-system application that wants to send raw video through RTP and wanting to use a video compression service (more specifically, a H.264 codec) followed by a video encapsulation service using MPEG, can use the following specification:

```
*:*[VIDEO(raw),PROTOCOL(rtp)]>>video_compression(H.264:targetres=320x240)
>>video_encapsulation(MPEG)>>192.168.1.2:80
```

During our implementation of this concept, we have considered a standard format for service specification. The initial service specification provided by one of the end-systems will only consist of the source and destination information. The properties of data (such as type, resolution, format etc) that are being transmitted are mentioned as semantics. The following example shows one of the service specifications used by us in our implementation.

```
10.2.0.2:6000(HDTV,1280x544,WMV)>>10.1.0.2:6004(H264,320x240,MPEG)
```

This service specification indicates the transfer of a Video file from the server (10.2.0.2:6000) to client (10.1.0.2:6004). The semantics at each end gives the kind of video file that can be played on that device. For the client to receive a video of the format as given in its semantics, the original video format available at the server needs to be converted. Thus, the initial service specification is provided to the Network Service Policy Manager (to be explained in the next section), which then introduces services if required based on policies defined and hence the service specification changes. In the above mentioned service specification, based on the policy rules defined, two services TranscodeResolution(320x240) and TranscodeFormat(WMvxMPEG) will be

introduced in the data path. These services will convert the initial video available at the server to a format required for the client. The updated service specification given by the Network Service Policy Manager is shown below:

```
10.2.0.2:6000(HDTV,1280x544,WMV)>>TranscodeResolution(320x240)
>>TranscodeFormat(WMVxMPEG)>>10.1.0.2:6004(H264,320x240,MPEG)
```

We expect that adoption of the next-generation Internet architectures will lead to standardization of network services and data semantics specifications leading to a flexible, extendable and standardized service and semantics specification framework.

5.2.2 Network Service Policy Management

The Network Service Policy Manager in the Service Controller is the primary module responsible for extraction of the semantic header from the request packet, verification of the semantics and service composition and enforcement of service policies. The figure 5.1 illustrates in detail the internal steps of the extended service socket API.

As mentioned earlier, here the end-system application includes the semantics of the data that it intends to send along with the services it needs in the service specification. The svc_request is the service request call that uses a lexer and a parser to interpret the service specification string, which is then translated into a request packet. It also prepares a Semantic Header that is appended to the request packet and then sent to the local service controller. The packet at the service controller is supplied to Network Service Policy Management module that extracts the Semantic Header and service composition as requested by the end-system. Depending on the semantics of the data and service composition, the Network Service Policy manager creates an OWL (Web Ontology Language) description of the services requested and the semantics of the data that will be sent by the end-system application. An integration of Python with Web Ontology Language, [2] is one of the suitable methods for implementing this.

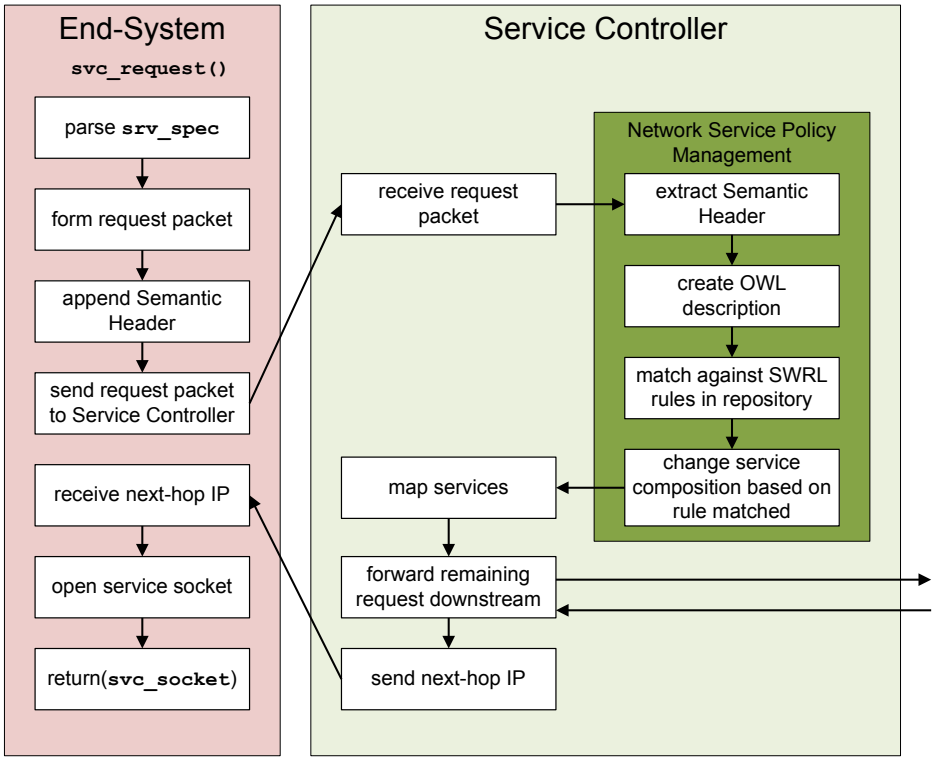

Figure 5.1. Extension to Service Socket API that processes and appends a semantic Header to the request packet.

It allows us to represent formal semantics in an XML-based syntax and automate the process. It also makes the syntax extremely easier to read for humans. OWL helps in representing the hierarchical constraints present among various services and data semantics in an XML like syntax. OWL provides us with a semantically rich language that is both extendable and flexible, which are very important with regards to altering service compositions in lieu of multi-party service policies. The various policy rules are represented using SWRL(Semantic web rule language).

OWL: Web Ontology Language (OWL) designed by the W3C Web Ontology Working Group, is a family of knowledge representation languages that is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for systems utilizing thesauri and other taxonomies. In practice, however, most of the expressiveness constraints placed on OWL Lite amount to little more than syntactic inconveniences: most of the constructs available in OWL DL can be built using complex combinations of OWL Lite features. Development of OWL Lite tools has thus proven almost as difficult as development of tools for OWL DL, and OWL Lite is not widely used.

- OWL DL was designed to provide the maximum expressiveness possible while retaining computational completeness (all conclusions are guaranteed to be completed), decidability (all computations will finish in finite time), and the availability of practical reasoning algorithms. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, number restrictions may not be placed upon properties, which are declared to be transitive). OWL DL is so named due to its correspondence with description logic, a field of research that has studied the logics that form the formal foundation of OWL.

- OWL Full is based on a different semantics from OWL Lite or OWL DL, and was designed to preserve some compatibility with RDF Schema. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right; this is not permitted in OWL DL. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for OWL Full.

Each of these sublanguages is a syntactic extension of its simple predecessor.

OWL is a component of the semantic web activity. This effort aims to make Web resources more readily accessible to automated processes by adding information about the resources that describe or provide Web content. As the Semantic Web is inherently distributed, OWL must allow for information to be gathered from distributed sources. This is partly done by allowing ontologies to be related, including explicitly importing information from other ontologies.

In addition, OWL makes an open world assumption. That is, descriptions of resources are not confined to a single file or scope. For example, while class c1 may be defined originally in ontology O1, it can be extended in other ontologies. The

consequences of these additional propositions about c1 are monotonic. New information cannot retract previous information. New information can be contradictory, but facts and entailments can only be added, never deleted. Hence, to write an ontology that can be interpreted unambiguously, we require a syntax and formal semantics for OWL. OWL is a vocabulary extension (RDF Semantics) of RDF. OWL can be represented using two types of syntaxes:

- Using RDF/XML documents: This is being used in general as OWL is part of the Semantic Web. This syntax is followed so that OWL can be an extension of RDF and RDF applications can parse OWL.
- Abstract Syntax: It is easier to read and write manually. It corresponds more closely to description logics and frames.

OWL can be used only by building Ontologies. Most of the elements of an OWL ontology concern classes, properties, instances of classes, and relationships between these instances. Ontology basically determines how the world (domain) should work and involves the following sequence of steps:

- determining the classes and properties in the domain.
- determining domains and ranges for properties.
- determining characteristics of classes.
- adding individuals and relationships as necessary.
- iterating until it is good enough.
- packaging all this into an ontology.

OWL's ability to express ontological information about instances appearing in multiple documents supports linking of data from diverse sources in a principled

way. The underlying semantics provides support for inferences over this data that may yield unexpected results. In particular, the ability to express equivalences using owl:sameAs can be used to state that seemingly different individuals are actually the same. Owl:InverseFunctionalProperty can also be used to link individuals together. For example, if a property such as "SocialSecurityNumber" is an owl:InverseFunctionalProperty, then two separate individuals could be inferred to be identical based on having the same value of that property. When individuals are determined to be the same by such means, information about them from different sources can be merged. This aggregation can be used to determine facts that are not directly represented in any one source.

The ability of the Semantic Web to link information from multiple sources is a desirable and powerful feature that can be used in many applications. However, the capability to merge data from multiple sources, combined with the inferential power of OWL, does have potential for abuse. Hence, users of OWL should be alert to the potential privacy implications.

SWRL: Semantic Web Rule Language (SWRL) is a proposal for a semantic web rules-language, combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language (Unary/Binary Datalog). SWRL includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL. It includes a model-theoretic semantics to provide the formal meaning of OWL ontologies including rules written in an abstract syntax, an XML syntax based on RuleML and the OWL XML Presentation syntax as well as an RDF concrete syntax based on the OWL RDF/XML exchange syntax.

The rules specified by SWRL will be in the form of an implication between an antecedent (body) and consequent (head). The intended meaning of this is that whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an empty consequent is treated as trivially false (i.e., not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction. Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values.

Rules offer users the ability to express certain logical relationships in a form suitable for machine processing. They are declarations like 'if P is true, then Q must also be true,' and for some applications they are easy for people to understand and efficient for machines to use in computation.

As a rule language for the Semantic Web, SWRL uses URIs to identify things, making it essentially compatible with RDF and OWL. For example, In RDF, an organization can express that a particular person is an employee and is also granted access to all internal documents. In SWRL, one can express the rule that all employees are granted access to internal documents. Given this rule and the fact that someone is an employee, a SWRL reasoner can conclude that the person is granted access.

SWRL is unique in being an extension of OWL DL, so that users of OWL DL can add rules to their ontologies and maintain clear semantics. Some rule systems offer meta-processing (rules about rules), and with the addition of OWL comes the possibility for new confusion in rules about OWL axioms and OWL axioms about rules; the design of SWRL 0.6 carefully steers clear of these potentially-confusing areas.

SWRL rules can be represented using the following set of syntaxes:

- Human Readable Syntax: In this syntax, rule has the form: antecedent=>consequent where both antecedent and consequent are conjunctions of atoms written `a1 ^`

... ^ an. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., ?x).

- XML Concrete Syntax: The XML Concrete Syntax is a combination of the OWL Web Ontology Language XML Presentation Syntax with the RuleML XML syntax.

- RDF Concrete Syntax: It is straightforward to provide an RDF concrete syntax for rules, but the presence of variables in rules goes beyond the RDF semantics. Translation from the XML Concrete Syntax to RDF/XML could be easily accomplished by extending the XSLT transformation for the OWL XML Presentation syntax.

Thus, in this work we used SWRL language as it is an extension of the semantics of OWL and provides the network operators with human-readable syntax and an easy to use method to specify precondition-action-postcondition and provisional execution rules. The OWL description is then matched with the SWRL rules. This is achieved by running a semantic reasoning engine that takes the OWL description and SWRL rules and determines the actions to be taken based on the policy that the matched rule describes. OWL is compatible and can be easily integrated with many semantic reasoning engines such as Bossam, Hoolet, Pellet etc. Depending on the action associated with the policy, new services can then be either added (service composition is changed) or other actions will be applied. Depending on the new service composition, the service controller will then make a routing and mapping decision to determine which of the services can be processed locally and which needs to be sent to the neighboring service controller. Once the path is set up, the service controller returns the information about the next hop to the end-system that requested the services. A service socket to the next hop is then opened and returned to the application.

We have used the pellet reasoner in this implementation. Pellet API provides various functionalities to see the species validation, check consistency of ontologies, classify the taxonomy, check entailments and answer a subset of RDQL queries(known as ABox queries in DL terminology). Pellet is an OWL DL reasoner based on the tableaux algorithms developed for expressive Description Logics. It supports the full expressivity OWL DL including reasoning about nominals (enumerated classes). Therefore, OWL constructs owl:oneOf and owl:hasValue can be used freely. Currently, Pellet is the first and only sound and complete DL reasoner that can handle this expressivity. Pellet ensures soundness and completeness by incorporating the recently developed decision procedure for SHOIQ(the expressivity of OWL-DL plus qualified cardinality restrictions in DL terminology). The main feature of this reasoner is the Conjunctive ABox query used for Query answering. This ABox query answering module is based on "rolling-up" technique. The algorithms devised optimize the query answering by changing how likely candidates for variables are found and tried. Exploiting the dependencies between different variable bindings helps us to reduce the total number of satisfiability tests thus speeding up the answer significantly.

Here, we have differentiated service policies based on their priorities. A service policy that is required for the proper working of a network is considered a high priority policy and thus is a 'forced' policy. For example, TCP is essential for reliable data transfer and hence is called as a forced policy. As against to this, a service like intrusion detection within an enterprise network is not really required if the parties involved are trusted. The same principle applies even when encrypted data is involved, as an intrusion detection service applied to on an encrypted payload does not yield any useful results. Such service policies are called as optional service policies.

CHAPTER 6

PROTOTYPE IMPLEMENTATION AND RESULTS

We have used a Cisco ISR (Integrated services router) as a service node of our network service architecture prototype. The Cisco ISR will run the Video transcoding application as a service.

6.1 Why Cisco ISR?

The Cisco Integrated Services Router (ISR) emphasizes the concept of an integrated system, which has the ability to tie together and run multiple value-added services such as voice, layer 2 switching, security and application acceleration. Integrated services can be hosted within the router operating system (IOS) or decoupled and hosted on modular application service modules. The Cisco ISR is provided with a new platform called AXP (Application Extensions Platform) that can host applications in a separate runtime environment with dedicated resources. This new platform is preferred due to the following reasons:

- It provides a predictable and constant set of resources to the host application (third party application). These resources (including CPU, memory, disk and network IO) are segmented in such a way that neither the host application nor the router can adversely affect the performance of the other.

- It provides an execution environment, which sufficiently separates the application space from the router space. Therefore, a crash of the host application cannot destabilize or crash the router.

- It provides an extensible and flexible platform for hosting third party applications. The platform can support multiple programming languages and hosting environments.

- It hosts a hardened Cisco Linux OS with virtualization and supports applications in programming languages like C, C++, Java , Tomcat and scripts like bash, perl and python.

- It provides robust debugging and troubleshooting facilities. The hosting environment further provides extensive logging and analysis tools to help support personnel determine the origin of the problem.

- It provides protection against unauthorized software. The platform provides mechanisms to ensure that only Cisco certified parties can install software on the AXP.

- It also provides facilities for third party applications to interact with IOS to produce advanced applications. These facilities consist of programmatic interface to modify IOS configuration, receive notification of events from IOS as well as the ability to access peripherals attached to the ISR.

These facilities make possible tighter integration of third party application in ISR thus increasing the attractiveness of the AXP platforms as hosting platform of choice, over server based solution.

In this work, we consider the implementation of a video distribution scenario and the service composition process for this is shown in the figure 6.1.

The figure 6.1 provides a general view of how the multiparty service policies overlook the service composition process. It illustrates how the network components such as sender/receivers (end-systems) and network service providers specify services according to their local and network wide service policies respectively, which are then used to compose services. We may have the following services in this scenario:

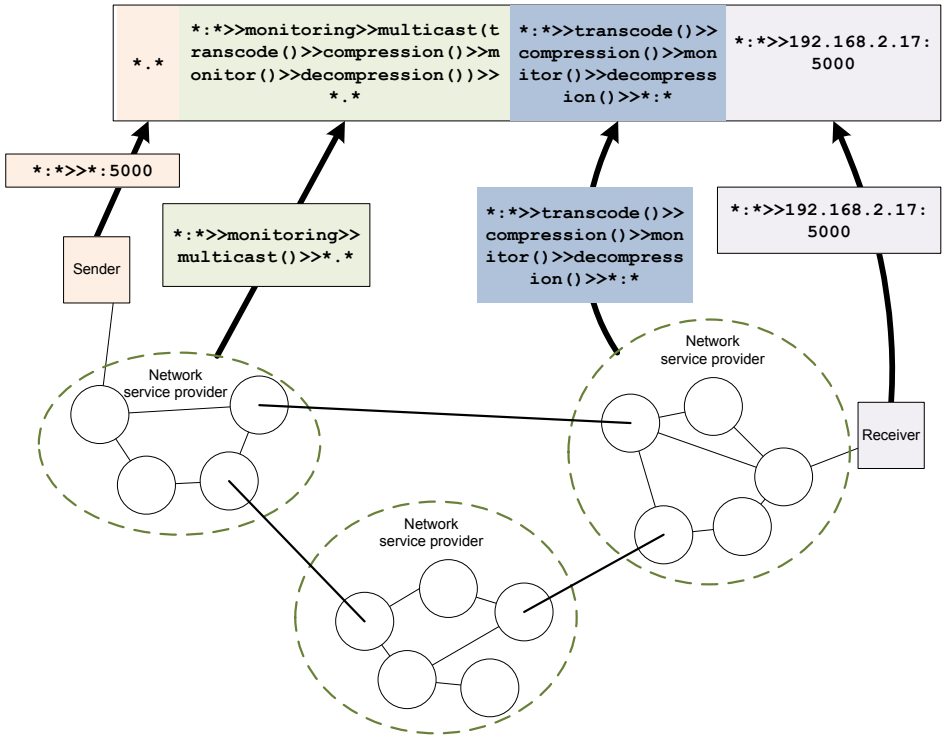

Figure 6.1. Service Composition Process.

- Transcoding: An end-system that receives a high-quality streaming video might not be capable of displaying it. For example, cellphone displays have a limitation on the resolution and bitrates. Thus, such an end-system may request a transcoding service conveying to the service controller the resolution, the streaming protocol used and the format of the video it is compatible with. The format of the video is determined by the codec installed on the end-system and is a representation of the data semantics, which enable the service controller to make informed decisions. Here, we use the Cisco ISR to act as a service node that runs the transcoding service.

- Monitoring: A network service provider might have a monitoring service policy on all video streaming connections to ensure Quality of service. Thus, even though the receiver has not requested a monitoring service in its service speci-

fication, the network service provider can choose to install one in the data path to the receiver.

- Multicast: The streaming video may be sent to multiple destinations as requested by the sender of the stream. In such a case, the service provider may choose to install separate monitoring services on all multicast paths to ensure QoS.

- Intrusion Detection: A service Provider may have a 'strict' intrusion detection service policy for all traffic through its network. However, applying an intrusion detection service for data known to be video streaming data results in wastage of resources and may result in degradation of service. In such cases, the service provider can take into account the semantics of the data being sent and drop the policy altogether for this type of traffic.

- Payload Compression/Decompression: To maintain QoS in networks affected by congestion, the monitoring service may trigger a service policy resulting in the network provider installing a payload compression/decompression service pair in the data path. In this case, it only makes sense to install the payload compression/decompression service pair between monitoring/transcoding services and the end-system. The services may be colocated. This may be taken care of by the reasoning engine that reasons over the OWL description of the service constraints and hierarchies, the current service specification and the SWRL rules describing service policies and determines the actions that need to be initiated. In this case, based on the service pair constraints between compression/decompression services, the reasoning engine may determine that exact nodes in the network where the new services may be placed.

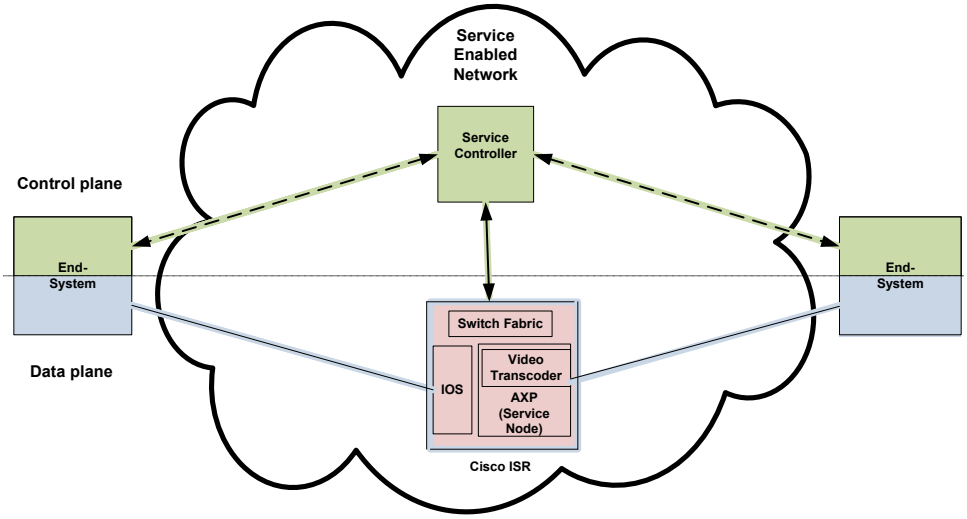

Figure 6.2. Prototype Setup of NSA.

6.2 Prototype Setup

Here we present our prototype of the network service architecture. The Cisco ISR acts as a service node that runs the video transcoding service. This service is only applied based on the network policy rules defined in the SWRL format. The figure 6.2 illustrates our prototype implementation.

6.2.1 Introducing Transcoding Services based on Policy Rules

There exist a wide range of client devices in this world and each one requires a different resolution of video to be played on it. For example, a laptop or a desktop can play a larger resolution(1280x544) HDTV video but a smaller client like a mobile device or a PDA can play only a smaller resolution(320x240) H264 video. Furthermore, even the video format requirements may vary from one client to other. One may require a WMV format, the other may require a MPEG format etc. Hence, to handle such different kinds of requirements of various types of client devices, here we introduce transcoding services based on client requirements and the available formats with the server based on policy rules.

We basically have a video file of different resolutions (such as 1280x544(HDTV), 640x480(H264), 320x240(H264)) and formats like WMV and MPEG at the server end. The policies are defined in the Network Service Policy Manager in such a way that depending on the video format,resolution and type required by the client device and those available with the server machine, new services will be introduced in the data path. the output resolution of the video delivered to the client varies depending on the client address. Thus, the transcoding services will be applied if the client has to receive a smaller resolution video and a different format than that of the server.

We have illustrated the above mentioned scenario using three different client addresses: 10.1.0.2:6004, 10.1.0.2:6005, 10.1.0.2:6006. The initial service specification is provided by the server(10.2.0.2:6000). As mentioned earlier, we have just followed a standard format with only server and client addresses and the data semantics such as properties of Video data being transferred (such as type, resolution and format). This initial service specification is then parsed to form a request packet and sent to the Service Controller. The Service Controller passes on this to the Network service Policy Manager, which then forms an OWL description of the request. The OWL formed is then matched against the predefined policy rules (defined using SWRL) with the help of pellet reasoner. The rule matched provides a solution of which service has to be applied next or, which node has to be included next in the data path. This process continues till the next node result is the same as the client address given in the initial service specification. The Network Service Policy Manager then provides with the updated service specification with all the intermediate nodes or services.

The above setup has been tested with 36 different scenarios varying video types, resolutions and formats both at server and client end. The chart in figure 6.3 shows the results of these 36 different scenarios.

The following are the notations used in figure 6.3:

```
R : denotes the resolution change service.
```

I/P \ O/P	H.264, 320x240, MPEG	H.264, 320x240, WMV	H.264, 640x480, MPEG	H.264, 640x480, WMV	HDTV, 1280x544, MPEG	HDTV, 1280x544, WMV
HDTV, 1280x544, WMV	R(1280x544 to 320x240) → F(WMV to MPEG)	R(1280x544 to 320x240)	R(1280x544 to 640x480) → F(WMV to MPEG)	R(1280x544 to 640x480)	F(WMV to MPEG)	-->
HDTV, 1280x544, MPEG	R(1280x544 to 320x240)	R(1280x544 to 320x240) → F(MPEG to WMV)	R(1280x544 to 640x480)	R(1280x544 to 640x480) → F(MPEG to WMV)	-->	F(MPEG to WMV)
H.264, 640x480, WMV	R(640x480 to 320x240) → F(WMV to MPEG)	R(640x480 to 320x240)	F(WMV to MPEG)	-->	X	X
H.264, 640x480, MPEG	R(640x480 to 320x240)	R(640x480 to 320x240) → F(MPEG to WMV)	-->	F(MPEG to WMV)	X	X
H.264, 320x240, WMV	F(WMV to MPEG)	-->	X	X	X	X
H.264, 320x240, MPEG	-->	F(MPEG to WMV)	X	X	X	X

Figure 6.3. Results showing the introduction of transcoder services in the data path.

```
F : denotes the format change service.

-->:  denotes normal forwarding of data without any services being
introduced.

X : denotes that no services exist for given service specification
and hence there will not be any connection established between
source and destination.
```

The following are some sample initial service specifications used to test the functionality of our system:

```
1) 10.2.0.2:6000(HDTV,1280x544,WMV)>>10.1.0.2:6004(H264,320x240,MPEG)

2) 10.2.0.2:6000(HDTV,1280x544,WMV)>>10.1.0.2:6005(H264,640x480,MPEG)

3) 10.2.0.2:6000(HDTV,1280x544,WMV)>>10.1.0.2:6006(HDTV,1280x544,WMV)
```

The OWL description of the above mentioned service specification created by the Network service Policy Manager is given figure 6.4 below:

In the OWL given in figure 6.4, the values of Source, Service,Type,Resolution,Format and Destination are filled from the initial service specification and the service result obtained from the initial service specification.

The Network Service Policy Manager has some policy rules defined corresponding to the above OWL description. These rules have been defined using the SWRL language as mentioned previously. The figure 6.5 shows the various policy rules in human readable syntax, defined by us to test our system:

When the OWL description of specification 1 is provided to the reasoner, rules 1, 2 and 4 will be matched introducing intermediate services TranscodeResolution(320x240) and TranscodeFormat(WMVxMPEG).In this case first the input video will be transcoded to a resolution of 320x240 and then it will be transcoded from WMV format to MPEG

```
<owl:Class rdf:ID="Source">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="Destination"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasdestination"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Node"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Node">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="Resolution"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasresolution"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasformat"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="Format"/>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="Type"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hastype"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasnextnode"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#Node"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:about="#Type">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Semantics"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Service">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
```

```
          <owl:Class rdf:about="#Destination"/>
        </owl:allValuesFrom>
        <owl:onProperty rdf:resource="#hasdestination"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Node"/>
  </owl:Class>
  <owl:Class rdf:about="#Format">
    <rdfs:subClassOf rdf:resource="#Semantics"/>
  </owl:Class>
  <owl:Class rdf:about="#Resolution">
    <rdfs:subClassOf rdf:resource="#Semantics"/>
  </owl:Class>
  <owl:Class rdf:about="#Destination">
    <rdfs:subClassOf rdf:resource="#Node"/>
  </owl:Class>
<Source rdf:ID="">
    <hasresolution rdf:resource="#"/>
    <hastype>
      <Type rdf:ID=""/>
    </hastype>
    <hasdestination>
      <Destination rdf:ID="">
        <hastype>
          <Type rdf:ID=""/>
        </hastype>
        <hasresolution>
          <Resolution rdf:ID=""/>
        </hasresolution>
        <hasformat rdf:resource="#"/>
      </Destination>
    </hasdestination>
    <hasformat rdf:resource="#"/>
  </Source>
  <Service rdf:ID="">
    <hasresolution rdf:resource="#"/>
    <hastype>
      <Type rdf:ID=""/>
    </hastype>
    <hasdestination>
      <Destination rdf:ID="">
        <hastype>
          <Type rdf:ID=""/>
        </hastype>
        <hasresolution>
          <Resolution rdf:ID=""/>
        </hasresolution>
        <hasformat rdf:resource="#"/>
      </Destination>
    </hasdestination>
    <hasformat rdf:resource="#"/>
  </Service>
```

Figure 6.4. OWL description of Service specification.

Rule 1:
Source(?x) hastype(?x, ?t1) hasresolution(?∧ ∧ x, 1280x544) ∧ hasformat(?x, ?f1) ∧ hasdestination(?x, ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, 320x240) ∧ hasformat(?y, ?f2) → hasnextnode(?x, TranscodeResolution(320x240))

Rule 2:
Service(TranscodeResolution(320x240)) ∧ hastype(TranscodeResolution(320x240), ?t1) ∧ hasresolution(TranscodeResolution(320x240), ?r1) ∧ hasformat(TranscodeResolution(320x240), WMV) ∧ hasdestination(TranscodeResolution(320x240), ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r2) ∧ hasformat(?y, MPEG)
→ hasnextnode(TranscodeResolution(320x240), TranscodeFormat(WMVxMPEG))

Rule 3:
Service(TranscodeResolution(320x240)) ∧ hastype(TranscodeResolution(320x240), ?t1) ∧ hasresolution(TranscodeResolution(320x240), ?r1) ∧ hasformat(TranscodeResolution(320x240), MPEG) ∧ hasdestination(TranscodeResolution(320x240), ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r2) ∧ hasformat(?y, WMV)
→ hasnextnode(TranscodeResolution(320x240), TranscodeFormat(MPEGxWMV))

Rule 4:
Service(TranscodeFormat(WMVxMPEG)) ∧ hastype(TranscodeFormat(WMVxMPEG), ?t1) ∧ hasresolution(TranscodeFormat(WMVxMPEG), ?r1) ∧ hasformat(TranscodeFormat(WMVxMPEG), ?f1) ∧ hasdestination(TranscodeFormat(WMVxMPEG), ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r2) ∧ hasformat(?y, ?f2)
→ hasnextnode(TranscodeFormat(WMVxMPEG), ?y)

Rule 5:
Service(TranscodeFormat(MPEGxWMV)) ∧ hastype(TranscodeFormat(MPEGxWMV), ?t1) ∧ hasresolution(TranscodeFormat(MPEGxWMV), ?r1) ∧ hasformat(TranscodeFormat(MPEGxWMV), ?f1) ∧ hasdestination(TranscodeFormat(MPEGxWMV), ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r2) ∧ hasformat(?y, ?f2)
→ hasnextnode(TranscodeFormat(MPEGxWMV), ?y)

Rule 6:
Service(TranscodeResolution(320x240)) ∧ hastype(TranscodeResolution(320x240), ?t1) ∧ hasresolution(TranscodeResolution(320x240), ?r1) ∧ hasformat(TranscodeResolution(320x240), ?f) ∧ hasdestination(TranscodeResolution(320x240), ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r2) ∧ hasformat(?y, ?f)
→ hasnextnode(TranscodeResolution(320x240), ?y)

Rule 7:
Source(?x) hastype(?x, ?t1) hasresolution(?∧ ∧ x, 1280x544) ∧ hasformat(?x, ?f1) ∧ hasdestination(?x, ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, 640x480) ∧ hasformat(?y, ?f2)
→ hasnextnode(?x, TranscodeResolution(640x480))

Rule 8:
Service(TranscodeResolution(640x480)) ∧ hastype(TranscodeResolution(640x480), ?t1) ∧
hasresolution(TranscodeResolution(640x480), ?r1) ∧ hasformat(TranscodeResolution(640x480),
WMV) ∧ hasdestination(TranscodeResolution(640x480), ?y) ∧ hastype(?y, ?t2) ∧
hasresolution(?y, ?r2) ∧ hasformat(?y, MPEG)
→ hasnextnode(TranscodeResolution(640x480), TranscodeFormat(WMVxMPEG))

Rule 9:
Service(TranscodeResolution(640x480)) ∧ hastype(TranscodeResolution(640x480), ?t1) ∧
hasresolution(TranscodeResolution(640x480), ?r1) ∧ hasformat(TranscodeResolution(640x480),
MPEG) ∧ hasdestination(TranscodeResolution(640x480), ?y) ∧ hastype(?y, ?t2) ∧
hasresolution(?y, ?r2) ∧ hasformat(?y, WMV)
→ hasnextnode(TranscodeResolution(640x480), TranscodeFormat(MPEGxWMV))

Rule 10:
Service(TranscodeResolution(640x480)) ∧ hastype(TranscodeResolution(640x480), ?t1) ∧
hasresolution(TranscodeResolution(640x480), ?r1) ∧ hasformat(TranscodeResolution(640x480), ?f)
∧ hasdestination(TranscodeResolution(640x480), ?y) ∧ hastype(?y, ?t2)
∧ hasresolution(?y, ?r2) ∧ hasformat(?y, ?f)
→ hasnextnode(TranscodeResolution(640x480), ?y)

Rule 11:
Source(?x) ∧ hastype(?x, ?t1) ∧ hasresolution(?x, ?r) ∧ hasformat(?x, MPEG) ∧
hasdestination(?x, ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r) ∧ hasformat(?y, WMV)
→ hasnextnode(?x, TranscodeFormat(MPEGxWMV))

Rule 12:
Source(?x) ∧ hastype(?x, ?t1) ∧ hasresolution(?x, ?r) ∧ hasformat(?x, WMV) ∧
hasdestination(?x, ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r) ∧ hasformat(?y, MPEG)
→ hasnextnode(?x, TranscodeFormat(WMVxMPEG))

Rule 13:
Source(?x) ∧ hastype(?x, ?t1) ∧ hasresolution(?x, ?r) ∧ hasformat(?x, ?f) ∧
hasdestination(?x, ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, ?r) ∧ hasformat(?y, ?f)
→ hasnextnode(?x, ?y)

Rule 14:
Source(?x) ∧ hastype(?x, ?t1) ∧ hasresolution(?x, 640x480) ∧ hasformat(?x, ?f1) ∧
hasdestination(?x, ?y) ∧ hastype(?y, ?t2) ∧ hasresolution(?y, 320x240) ∧ hasformat(?y, ?f2)
→ hasnextnode(?x, TranscodeResolution(320x240))

Rule 15:
Service(TranscodeResolution(640x480)) hastype(T ∧ ranscodeResolution(640x480), ?t1) ∧
hasresolution(TranscodeResolution(640x480), ?r1) ∧ hasformat(TranscodeResolution(640x480), ?f)
∧ hasdestination(TranscodeResolution(640x480), ?y) ∧ hastype(?y, ?t2)
∧ hasresolution(?y, ?r2) ∧ hasformat(?y, ?f)
→ hasnextnode(TranscodeResolution(640x480), ?y)

Figure 6.5. Policy rules defined in SWRL.

format. The first service specification updated by the Network Service Policy Manager is given below:

```
10.2.0.2:6000(HDTV,1280x544,WMV)>>TranscodeResolution(320x240)
>>TranscodeFormat(WMVxMPEG)>>10.1.0.2:6004(H264,320x240,MPEG)
```

Similarly, When the OWL description of specification 2 is provided to the reasoner, rules 7, 8 and 4 will be matched introducing intermediate services TranscodeResolution(640x480) and TranscodeFormat(WMVxMPEG).Here, the input video will be first transcoded to a resolution of 640x480 and then it will be transcoded from WMV format to MPEG format. The service specification updated by the Network Service Policy Manager in this case is given below:

```
10.2.0.2:6000(HDTV,1280x544,WMV)>>TranscodeResolution(640x480)
>>TranscodeFormat(WMVxMPEG)>>10.1.0.2:6005(H264,640x480,MPEG)
```

When the OWL description of specification 3 is provided to the reasoner, rule 13 will be matched. In this case, no additional service will be introduced and hence the input video will be delivered to the client without any transcoding. The service specification given by the Network Service Policy Manager in this case is the same as its initial specification.

```
10.2.0.2:6000(HDTV,1280x544,WMV)>>10.1.0.2:6006(HDTV,1280x544,WMV)
```

The figure 6.6 illustrates the space-time description of the above explained setup.

6.3 Results

We have set up a prototype of the NSA (Network service architecture) using the Cisco ISR as the service node. This set up has a service node implementing the video transcoding application. The figure 6.7 illustrates this setup. We have considered two

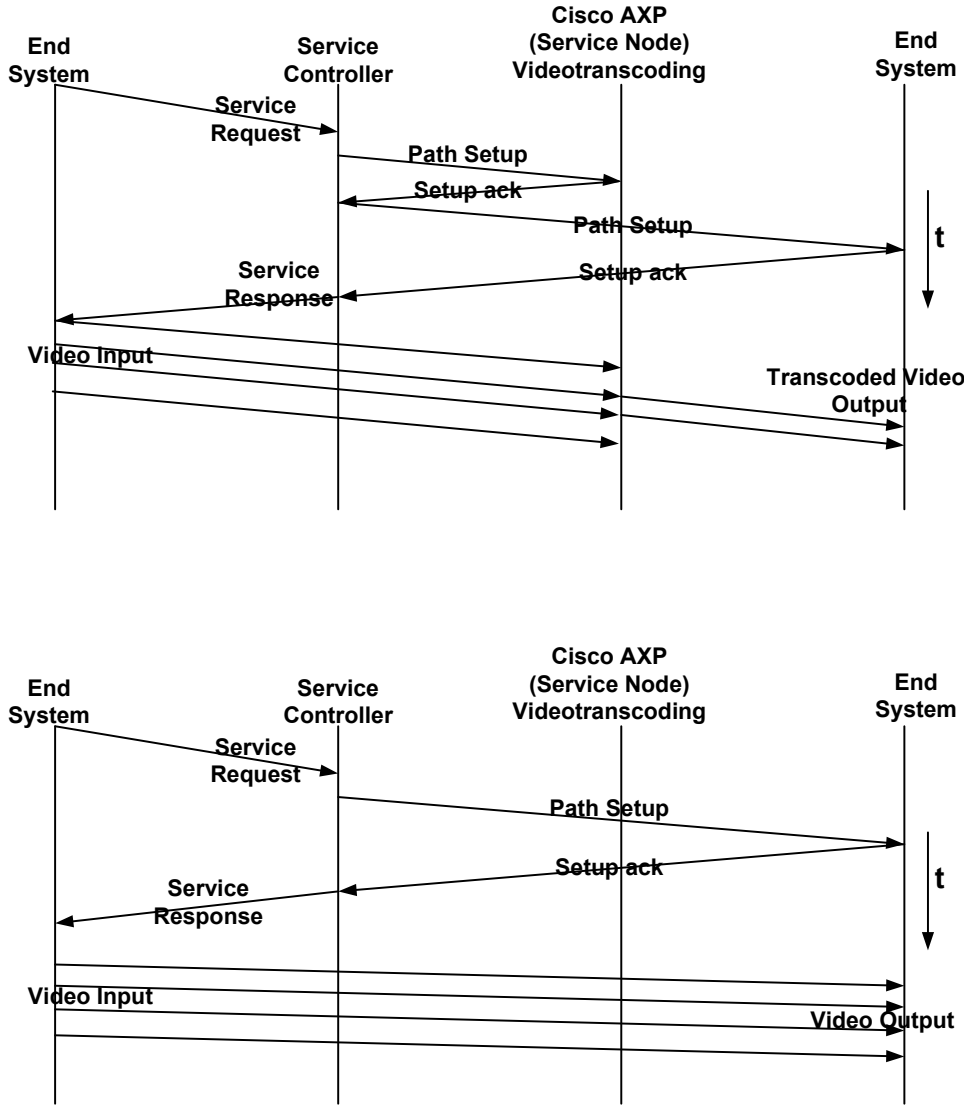

Figure 6.6. Space-time description of the service specification example with and without transcoding service enabled.

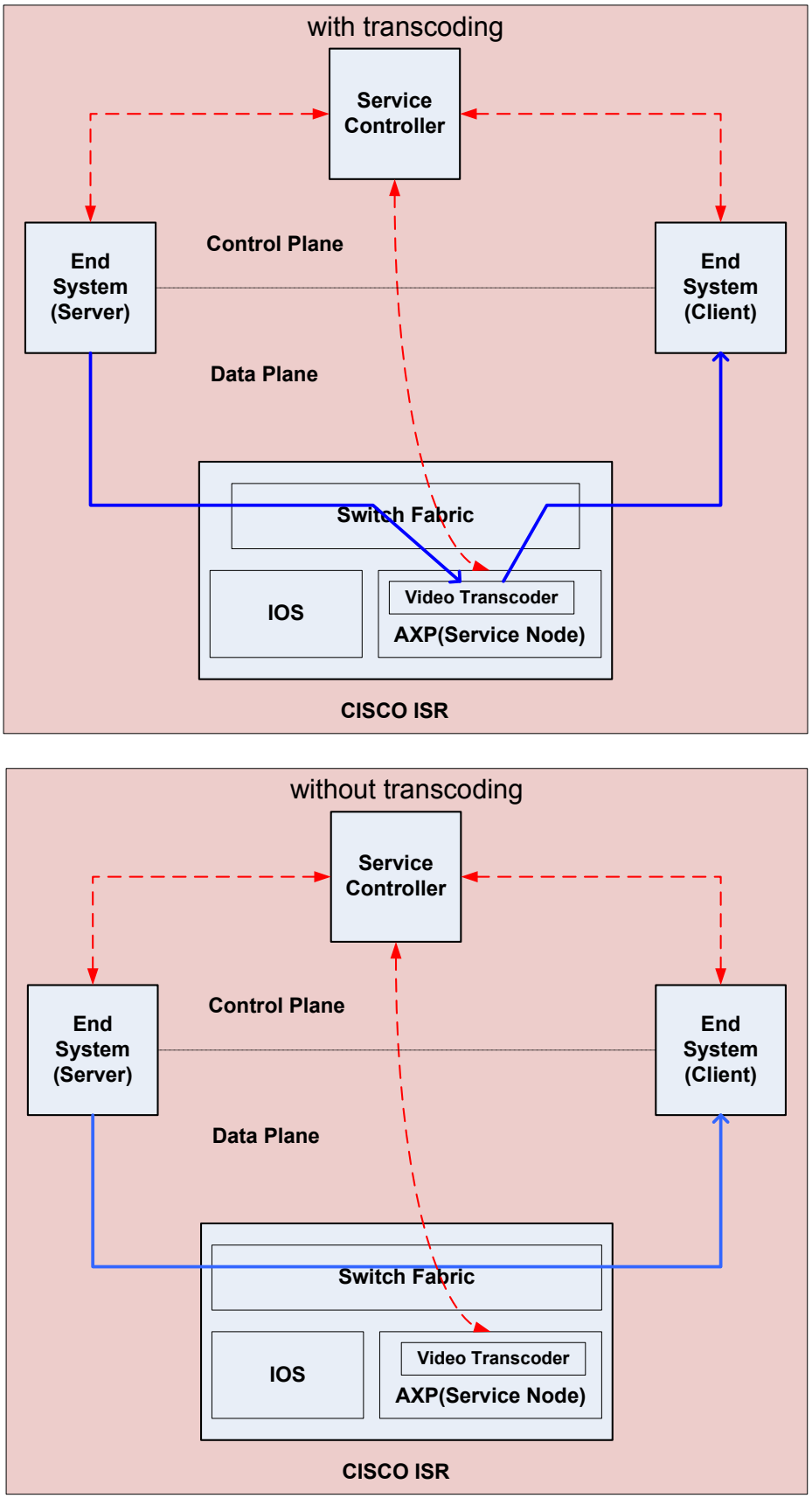

Figure 6.7. Prototype Implementation of NSA on Cisco ISR with and without transcoding service enabled.

scenarios, first with the transcoding service disabled and then with the transcoding service enabled depending on the policy rules.

When the Network Service Policy Manager does not introduce a transcoding service, which is based on the client requirements in the initial specification given, the transcoding service is not applied and the packets are simply forwarded to the Client. However, when the Network Service Policy Manager introduces a transcoding service the packets are rerouted to the AXP, which runs the Videotranscoding service and hence converts to a different resolution based on the resolution the client device wanted. Therefore, as per the policy rules defined previously if the client requires a Video of resolution 320x240, the video is transcoded to a resolution of 320x240, if the client requires a Video of resolution 640x480, it is transcoded to a resolution of 640x480 and if the client requires the same resolution as with the server, the video is streamed as it is without transcoding (i.e. without any change in the resolution). The figure 6.8 shows the performance of these three scenarios.

Here, we have transcoded a video from a resolution of 1280x544 to 320x240 and 640x480. It is clearly evident from the figure that the data rate is significantly reduced due to transcoder (i.e. in the case when transcoding service is introduced, the input shows a higher data-rate and the output shows a lower data-rate while when there is no transcoding the input and output data-rates are almost equal). Thus, the transcoded video stream with a smaller resolution can be displayed on a low end client. A noteworthy point here is that the sender is unaware of the transcoding process as the transcoder service is only introduced based upon the policies defined at the Network Service Policy Manager part of the Service Conroller (as explained earlier).

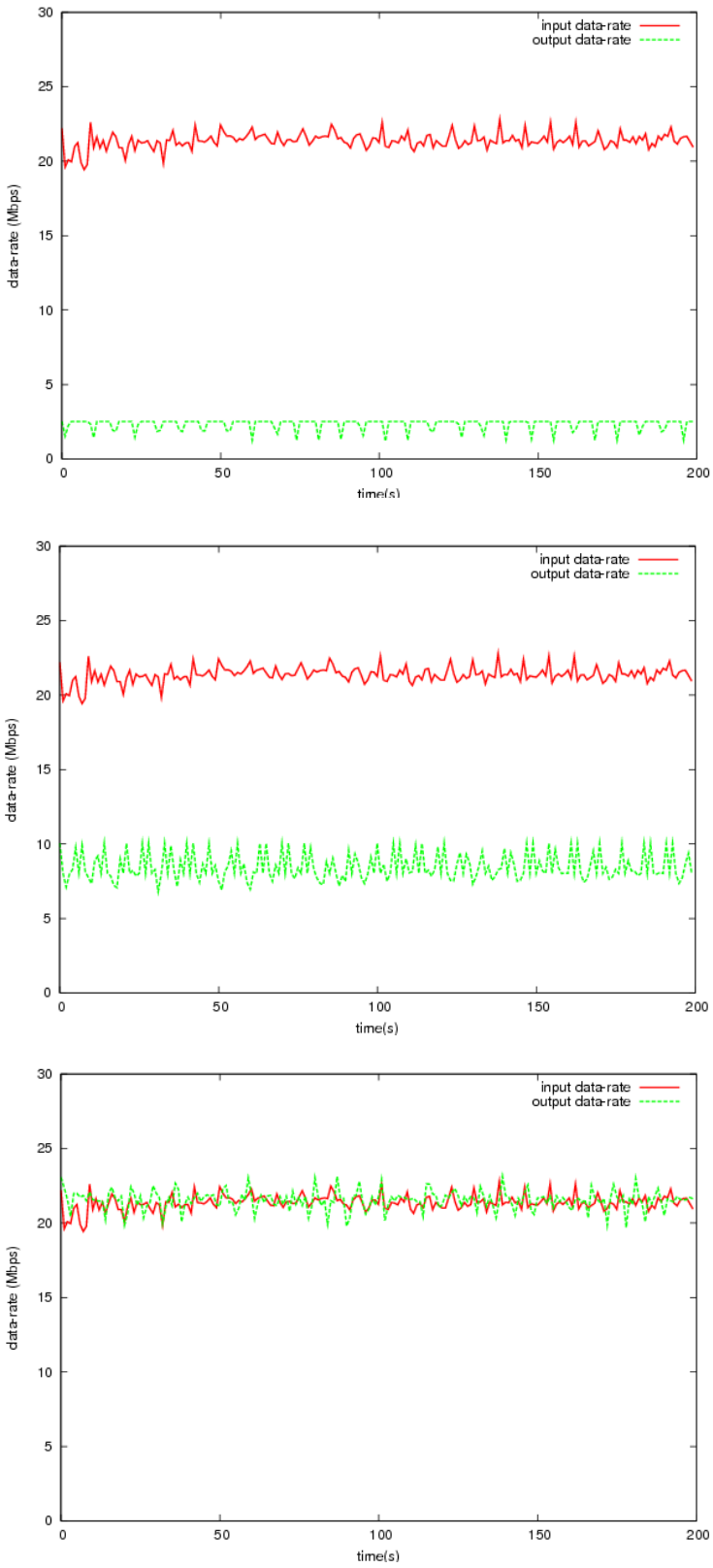

Figure 6.8. Data-rates when a video of resolution 1280x544 is transcoded to 320x240, 640x480 and then no transcoding

CHAPTER 7

CONCLUSION AND SUMMARY

The main theme of this thesis was implementing the prototype of a network service architecture with multi-party service composition and service placement based on network policies. This was implemented using the semantics of the data sent by the end-system along with the service request.In our prototype setup we have implemented the Videotranscoding service on a Cisco AXP module. The multi-party service specification framework introduced extends the service socket API [20]. We have thus leveraged the use of semantics of data in multiparty service specifications. Our multiparty service specification framework allows end-system applications to easily utilize network services without exposing them to the complexities of connection setup and network policies. Furthermore, we believe that this approach represents an important step towards making highly dynamic and flexible communication configurations an integral part of the next-generation Internet.

7.1 Contribution

My contributions to this work can be summarized as:

- Extended the service specification framework and service sockets API [20] to perform all the functionalities as shown in figure 5.1.
- Implemented the enforcement of the network policies based on the data semantics sent by the end-system. This is done using W3C Web Ontology Language(OWL) [3].

- Implemented a repository of network rules that are based on policies in multi-party service composition for the purpose of network management. This is done using W3C Semantic Web Rule Language (SWRL) [1].

- Implemented the Videotranscoding application on Cisco AXP, which acts as a service node in our prototype of Network Service Architecture. This service is enabled based on the network rules matched.

7.2 Future Work

There is however still scope for improvements in the future. Here we have implemented our prototype setup by considering only one service(Videotranscoding). This can be further extended by introducing additional services for different types of data. Correspondingly, we can also define a large number of rules to impose different kinds of policies.

BIBLIOGRAPHY

[1] *Semantic Web Rule Language (SWRL).* `http://www.w3.org/Submission/SWRL/`.

[2] *Seth: Semantic Python.* `http://seth-scripting.sourceforge.net/`.

[3] *Web Ontology Language (OWL).* `http://www.w3.org/2004/OWL/`.

[4] Anderson, Thomas, Peterson, Larry, Shenker, Scott, and Turner, Jonathan. Overcoming the Internet impasse through virtualization. *Computer 38*, 4 (Apr. 2005), 34–41.

[5] Baldine, Ilia, Vellala, Manoj, Wang, Anjing, Rouskas, George, Dutta, Rudra, and Stevenson, Daniel. A unified software architecture to enable cross-layer design in the future internet. In *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Honolulu, HI, Aug. 2007).

[6] Barbir, Abbie, Reinaldo, Penno, Chen, Robin, Hofmann, Markus, and Hilarie, Orman. An architecture for open pluggable edge services (OPES). RFC 3835, Network Working Group, Aug. 2004.

[7] Bhatti, Nina T., and Schlichting, Richard D. A system for constructing configurable high-level protocols. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication* (Cambridge, MA, Aug. 1995), pp. 138–150.

[8] Braden, Robert, Faber, Ted, and Handley, Mark. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Computer Communication Review 33*, 1 (Jan. 2003), 17–22.

[9] Casado, Martin, Garfinkel, Tal, Akella, Aditya, Freedman, Michael J., McKeown, Dan Bonehand Nick, and Shenker, Scott. SANE: A protection architecture for enterprise networks. In *15th USENIX Security Symposium* (Vancouver, Canada, Aug. 2006), pp. 137–151.

[10] Egevang, Kjeld Borch, and Francis, Paul. The IP network address translator (NAT). RFC 1631, Network Working Group, May 1994.

[11] Feldmann, Anja. Internet clean-slate design: what and why? *SIGCOMM Computer Communication Review 37*, 3 (July 2007), 59–64.

[12] Floyd, Sally, and Jacobson, Van. Random early detection (RED) gateways for congestion avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (Aug. 1993), 397–413.

[13] Ganapathy, Sivakumar, and Wolf, Tilman. Design of a network service architecture. In *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Honolulu, HI, Aug. 2007), pp. 754–759.

[14] Hutchinson, Norman C., and Peterson, Larry L. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering 17*, 1 (Jan. 1991), 64–76.

[15] IETF. *Open Pluggable Edge Services*, 2003. `http://standards.nortelnetworks.com/opes/index.htm/`.

[16] Keller, Ralph, Ramamirtham, Jeyashankher, Wolf, Tilman, and Plattner, Bernhard. Active pipes: Program composition for programmable networks. In *Proc. of the 2001 IEEE Conference on Military Communications (MILCOM)* (McLean, VA, Oct. 2001), pp. 962–966.

[17] Mogul, Jeffrey C. Simple and flexible datagram access controls for UNIX-based gateways. In *USENIX Conference Proceedings* (Baltimore, MD, June 1989), pp. 203–221.

[18] Rudra Dutta, George N. Rouskas, Baldine, Ilia, Bragg, Arnold, and Stevenson, Dan. The SILO architecture for services integration, control, and optimization for the future internet. In *Proc. of IEEE International Conference on Communications (ICC)* (Glasgow, Scotland, June 2007), pp. 1899–1904.

[19] Ruf, Lukas, Farkas, Karoly, Hug, Hanspeter, and Plattner, Bernhard. Network services on service extensible routers. In *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)* (Sophia Antipolis, France, Nov. 2005).

[20] Shanbhag, S., and Wolf, T. Implementation of end-to-end abstractions in a network service architecture. In *Fourth Conference on emerging Networking EXperiments and Technologies (CoNEXT)* (Madrid, Spain, Dec. 2008).

[21] Snort. *The Open Source Network Intrusion Detection System*, 2004. `http://www.snort.org`.

[22] Tennenhouse, David L., and Wetherall, David J. Towards an active network architecture. *ACM SIGCOMM Computer Communication Review 26*, 2 (Apr. 1996), 5–18.

[23] Wolf, Tilman. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Arlington, VA, Oct. 2006), pp. 79–86.