Policy-driven Security Management for Gateway-Oriented Reconfigurable Ecosystems

by

Clinton Dsouza

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved January 2015 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Chair
Adam Doupé
Partha Dasgupta

ARIZONA STATE UNIVERSITY

May 2015

ABSTRACT

With the increasing user demand for low latency, elastic provisioning of computing resources coupled with ubiquitous and on-demand access to real-time data, cloud computing has emerged as a popular computing paradigm to meet growing user demands (Daly (2013)). However, with the introduction and rising use of wearable technology and evolving uses of smart-phones, the concept of Internet of Things (IoT) has become a prevailing notion in the currently growing technology industry. Cisco Inc. has projected a data creation of approximately 403 Zetabytes (ZB) by 2018 (Bradley (2013)). The combination of bringing benign devices and connecting them to the web has resulted in exploding service and data aggregation requirements, thus requiring a new and innovative computing platform. This platform should have the capability to provide robust real-time data analytics and resource provisioning to clients, such as IoT users, on-demand. Such a computation model would need to function at the edge-of-the-network, forming a bridge between the large cloud data centers and the distributed connected devices.

This research expands on the notion of bringing computational power to the edge-of-the-network, and then integrating it with the cloud computing paradigm whilst providing services to diverse IoT-based applications. This expansion is achieved through the establishment of a new computing model that serves as a platform for IoT-based devices to communicate with services in real-time. We name this paradigm as Gateway-Oriented Reconfigurable Ecosystem (GORE) computing. Finally, this thesis proposes and discusses the development of a policy management framework for accommodating our proposed computational paradigm. The policy framework is designed to serve both the hosted applications and the GORE paradigm by enabling them to function more efficiently. The goal of the framework is to ensure uninterrupted communication and service delivery between users and their applications.

To my parents

ACKNOWLEDGMENTS

My journey through my Computer Science degree both B.S and M.S has been an exciting one which has played a major role in my career and life, especially in enhancing my knowledge and experience in this field of study. Having had the opportunity to work with Dr. Gail-Joon Ahn for two years has been an insightful experience, and I am utmost grateful to him for having given me the opportunity to work on cutting edge research projects at the laboratory for Security Engineering for Future Computing (SEFCOM). Dr. Ahn has been a constant source of motivation through my graduate career at Arizona State University, and has contributed significantly towards my abilities in reasoning, approaching and solving research problems impacting the society as a whole. I would like to extend my sincere gratitude to Dr. Partha Dasgupta and Dr. Adam Doupé for serving on my committee and providing their valuable feedback on my thesis.

My experience at the SEFCOM lab has provided me with great opportunities in my professional career, and has enabled me to connect with brilliant and innovative individuals who have always been there to provide valuable inputs during my research work. I would specially like to express my gratitude to Marthony Taguinod, who worked with me in the development of our test-bed for Cisco Inc. In addition to my committee and the SEFCOM lab members, I would also like to thank Cisco Inc. and more specifically Dr. Rodolfo Milito, for their support in the Fog Computing project and their valuable input during the development of our test-bed. Finally, and most importantly, I would like to extend my sincere love and regards to my parents for being a constant source of motivation and support throughout my academic career.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Recent technology developments and device releases, most notably those utilizing the concept of Internet of Things (IoT) have gained considerable popularity from both academia and industry professionals. The demand for these devices is attributed to the fact that they can connect to the Internet and among themselves, and they can share data in real-time. Although data can be shared in real-time or near real-time among devices, it is challenging to achieve the same when devices are required to connect to remote servers and relate information between services hosted in computing paradigms such as cloud computing. Additionally, when a large number of these devices attempt to request information and services (such as data services, traffic services, or health-care services) from a single component, latency becomes a pressing and critical issue that must be addressed.

1.1 Internet of Things and Related Concepts

The term Internet of Thing was first introduced by Kevin Ashton in 1997 when he became interested in using RFID to help manage P&G's supply chain (McHugh (2004)). However the concept of IoT was being put into practice long before the term was coined. Traces of making devices smarter and giving them the capability to connect to the Internet date back to 1993 when the computer laboratory at the University of Cambridge created the Trojan Room Coffee Pot (Stafford-Fraser (1993)). Fraser and his team converted ("hacked") their coffee pot to send images of the coffee level in their pot to the web.

Based on device functionality and utilization the Internet of Things (IoT) can be

defined as a network of physical objects with the capability of communicating with associated smart devices either directly or via the Internet thus enabling the sharing and migration of data when required (Evans (2012)). This definition of IoT takes into account devices with the capability of connecting to the web, however the definition fails to take into account the inclusion of a human variable.

The data shared by IoT enabled devices is not just simple pieces of information, but, due to people owning these devices, the information is more personal such as user credentials, location information, biometric data, health-care related statistics, and more, which further requires caution in securing and maintaining the integrity of data. This inclusion of a human variable has given rise to a global concept termed as the Internet of Everything (IoE). Cisco defines IoE as "bringing together people, process, data and things to make networked connection more relevant and valuable". The concept of IoE encompasses every smart connected device, including personal devices such as smart-phones and tablets, thus making IoT a significant component of the larger IoE market (Evans (2012)).

Such connectivity and share-ability of device and information enables the proliferation of connected objects, and in turn these objects create a data explosion which comes from billions of devices located around the world. However, unless these distributed devices can communicate with each other, all the data is meaningless and may eventually result in wastage of resources and higher storage costs (Mora (2014)).

1.2 Cloud Computing

Cloud computing has been developed along a number of lines. Increase in bandwidth over the years has allowed for larger provisioning of resources including computing, networking, and storage in distributive environments. One of the first major milestones in the area of cloud is the establishment of salesforce.com in 1999. Sales-

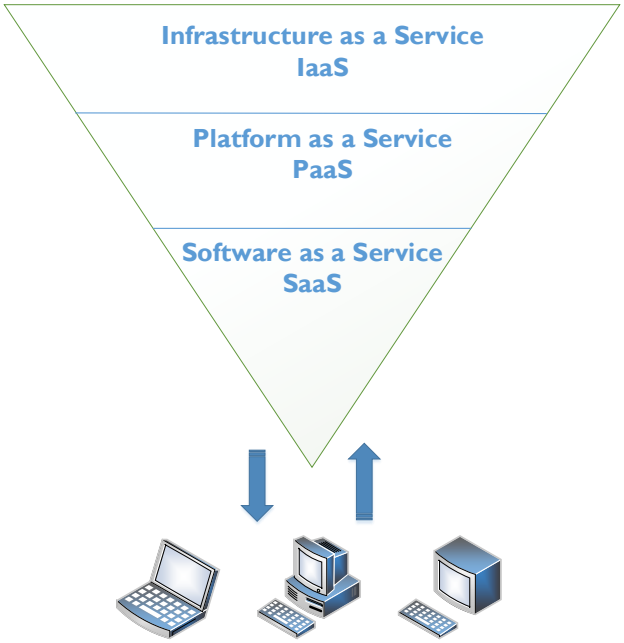

Figure 1.1: Cloud Infrastructure.

force opened a new era of cloud computing, which evolved to an enterprise model and paved the way for companies like Amazon EC2 and Windows Azure cloud services to provide enterprise-level service to clients on demand.

Current cloud computing infrastructure consists of three concrete layers as shown in Fig 1.1:

- Software as a Service (SaaS): The application layer of the cloud computing paradigm designed to host client-centric applications that can be accessed by users worldwide at anytime.

- Platform as a Service (PaaS): The delivery layer designed to include platform tools such as application execution environments, operating systems, databases, and web-servers.

- Infrastructure as a Service (IaaS): The physical entity of the cloud paradigm,

which often involves a virtual component such as virtual machines running on hypervisors. The IaaS is designed to provide the physical components required to support the PaaS and SaaS layers. This includes server data centers, data storages, hypervisors, networking routers, switches and so-on.

The maturity of the cloud computing model has reduced the overall cost of users owning and managing private data centers. The cloud model is designed to deliver computing as a service rather than a product. This business model has given rise to increasing user demand for the resource utilization including computing, networking, and storage. The cloud infrastructure itself supports a wide array of use-cases. These use-cases could range from simple data storage to large scale enterprise application support. Given the elasticity and flexibility of the cloud model as evident from providers such as Amazon and Microsoft, developers believe that cloud computing could support the large quantity of data generated from the smart connected devices also referred to as IoT.

With the current cloud infrastructure set-up and usage model, IoTs communicate directly with the virtualized instance through a Software-as-a-Service API which is configured by an individual tenant renting resources from a vendor. Fig 1.2 illustrates an overview of the current cloud usage model and its interaction with IoT based devices. In addition to its usage model, the cloud infrastructure is designed to provide centralized intelligence services to clients on demand. However, with the growing data storage demand, retrieval of data becomes a challenge, due to resource centralization.

1.3 IoT and the Big Data Crisis

The earliest records of large data collection date back to the 1880's when the first United States census was recorded (Bureau (1990)). The census took eight years to

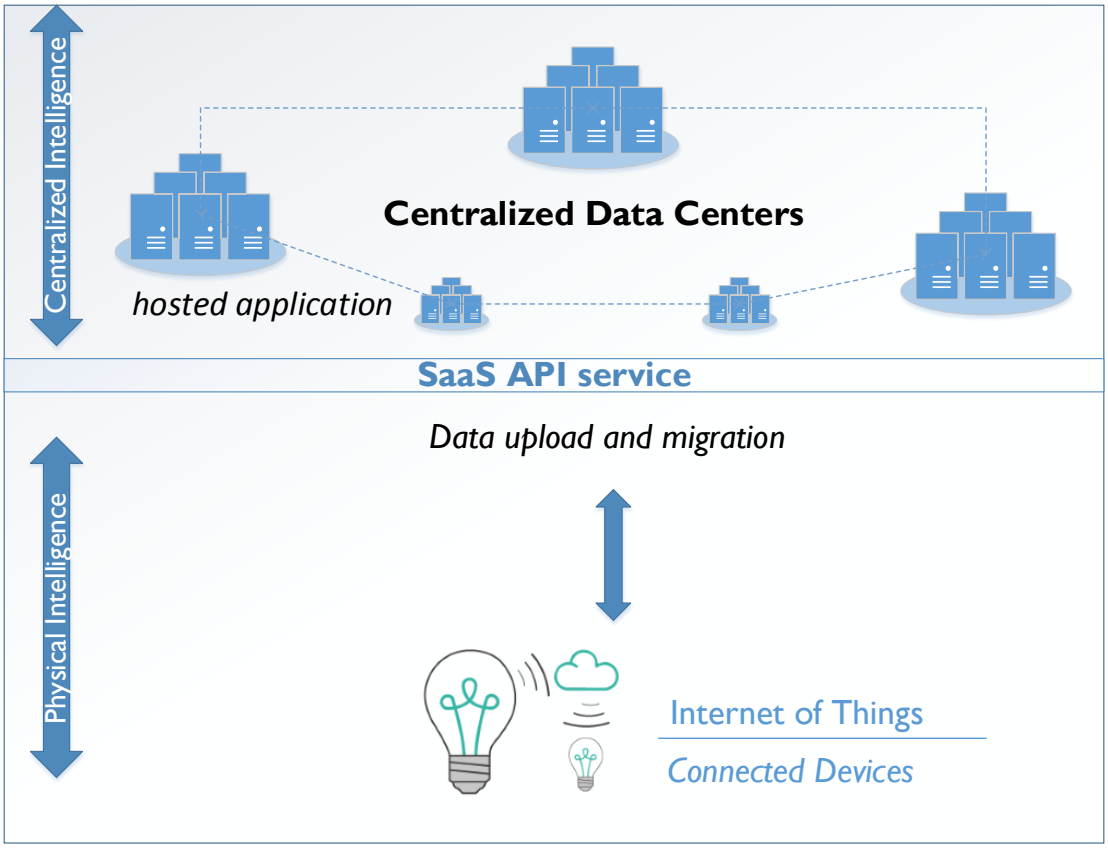

Figure 1.2: Interaction Between Cloud and IoT.

tabulate and estimations made stated that the 1890 census would take more than 10 years using the then-available methods. The 1930's ushered in a population boom in the US bringing along with it a large quantity of data. This data included personal identifiers such as social security numbers, user information and statistics, household information, and more. The data recorded was for general research and demanded more organized and thorough record keeping (Winshuttle (2014)). The arrival of the 1940's brought with it the first warning of the data storage and retrieval problem. Rider estimates "the Yale Library in 2040 would have approximate 200,000,000 volumes which will occupy 6,000 miles of shelves requiring a cataloging staff of over six

thousand persons" (Press (2013) and Bell (2008)).

KPMG, one of the largest professional services company in the world and one of the big four auditors of financial service in the United States, estimated a 30 percent digital data explosion from 2011 to 2012 with approximately 1.8 ZB in store for 2011 (International (2012)). This data storage is further estimated to increase to 35 ZB by 2020. This growing data statistics take into account all the data generated from various industries such as transportation, health, investment banking, and others. How this data is generated is an interesting question to delve into. While portions of the data will be generated through direct user input and interaction with systems, it is believed that automated data generation will play large role in contributing to the data growth. This automated data generation could partially be a result of systems monitoring user interaction or even systems analyzing and generating analytics based on user behavior patterns. A simple example would be the health care industry: a hospital room has numerous sensors and data collecting equipments that monitor a patients' vital stats. If this information is uploaded to servers, they are considered to be collecting and storing data. Given the widespread use of smart connected devices, and the sensors that could potentially be embedded into them, these smart devices also known as IoT will play a big role in the expected data explosion by 2020.

In 2013, the United Nation Economic Commission for Europe (UNECE) created a temporary task team to evaluate the usage of Big Data for official statistics, identify priority actions, and formulate project proposals (UNECE (2013)). Fig 1.3 illustrates the obtained data growth projected by this team from various European countries. The graph itself projects the explosive data growth expected by 2020 and shows the Big Data problem we face.

As we discussed earlier the growing usage of the term IoT and the addition of a human variable have given rise to the term IoE. It has estimated to capture a $14.4

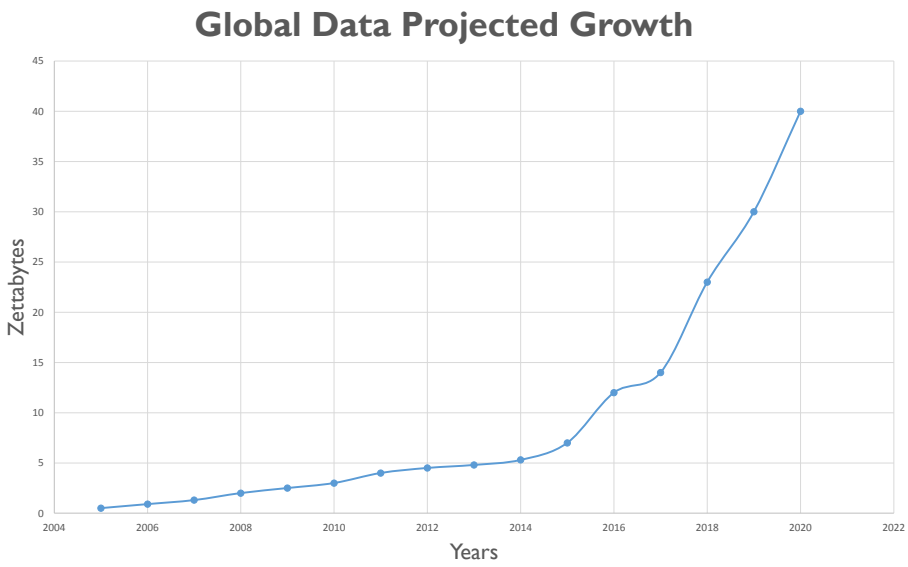

Figure 1.3: Big Data Growth (UNECE (2013)).

trillion market value with its usage (Cisco (2013)). Given the large cost and the even bigger data storage requirement raises concerns of connecting this data to the user and making sense of the large collection of data associated with a single user.

The solution to the Big Data problem is not simple, however successful industrial solutions involve cloud archival systems, such as QlikView and RainStor ((Qlik.com (2014) and RainStor (2013)), that enable storage of large quantities of data for historical analysis and usage. However, when it comes to querying, analyzing, and implementing functionalities to efficiently utilize this data, these tools do not provide a comprehensive solution as they still depend heavily on the cloud model for computation. As discussed in Chapter 1.2, the cloud computing model serves as a centralized resource center for application and data storage, however given the quantity of data being stored, we believe a more distributed model is required for efficient data aggregation.

1.4 Support for Cloud Computing

Current computational models such as cloud computing and service oriented computing cannot account for and handle the huge data load that will be generated by the boom of IoT-based devices, and thus require an evolved computing paradigm with the capability of communicating more closely with physical smart devices, a paradigm that will behave as a gateway supporting interoperability among heterogeneous systems and networks. Such a paradigm would extend the cloud services to the edge of the network. For purposes of this work, edge-network refers to a layer or interface that sits between the physical world and the virtual components (SaaS or PaaS layers in cloud).

With the capability to sit at the edge of the network and provision resources on-demand, this envisioned computing paradigm has the capability to provide real-time data services to customers as well as perform dynamic data aggregation based on customer provided data. One such computational model has been recently proposed and formulated as a basis of this research, called the Fog Computing paradigm (Bonomi *et al.* (2012)). This thesis strives to achieve a computing model similar to the Fog model, yet more innovative, unique, and robust. Therefore, this research will describe the architecture for a *Gateway-Based Computational paradigm* highlighting prominent concepts which are believed to provide the robustness and interoperability desired for the successful interaction of IoTs.

To enable efficient, on-demand and real time support between smart devices and their connecting services, we believe a new computing paradigm would assist in supporting mobility and geo-distribution in combining with location awareness and low-latency communication. Such a paradigm would function as a gateway to a larger cloud computation model and would need to provision resources at the edge of the

network wherein the virtual and physical components of technology interact to request services. We refer to such a computational paradigm as a gateway-based cyber-physical paradigm wherein access to resources and services are dynamically provisioned by a platform that intelligently detects, senses, and aggregates data from services based on client demand. The model thus envisioned from this paradigm is called *GORE: Gateway-Oriented Reconfigurable Ecosystems*. Additionally, provisioning of services is not performed entirely by a virtual instance as in the Cloud Computing model, but can be performed by physical components with access to a virtual instance or a component that is virtualized to contain multiple virtual instances.

With the high density of IoTs being introduced into enterprises, including manufacturing floors, health-care devices, smart transportation systems, and smart grids, the realization of middle-ware layer computing or edge computing represents a futuristic and novel computational model which provides consumers with the capability to request and utilize data both in real-time as well as over a distributive and heterogeneous network whilst still having the capability to store large data into cloud data centers. However, with any new technology or computational paradigm, security is of importance and high concern, thus raising questions about data and communication security as well as data migration and access.

The term *security* covers a wide array of topics covering user identification and authentication up to data validation, analysis, and encryption. While security is a broadly used terminology, for purposes of this work security is focused around authentication, authorization, and validation of components in a system. This work further delves into these three entities of security and focuses on a single security entity that is considered vital to any core functionality of a system: *Policy Management*. This work is thus constrained to analyzing and researching the need for a policy management module in a distributed paradigm such as edge computing so as to enable

secure collaboration among cyber-physical systems and their interacting users.

Given the density of devices in a distributed environment, managing secure communication between disparate systems and users becomes a challenge, especially with the need for real-time communication. Supporting collaboration among IoTs is a gateway-based computational paradigm which in itself is a complex architecture due to its location at the edge of the network. The complexity of the GORE model involves multiple layered infrastructure including an orchestration layer wherein all the intelligent data aggregation occurs. Due to the high volume of moving variables in such a paradigm there is a need for a fine grained policy management framework module to ensure secure collaboration and secure provisioning of services and resources to clients in real-time.

This work will focus on the GORE architecture and will further explore and research the security infrastructure of such a framework from a policy management perspective. The end goal of this work is to ensure seamless communication and secure collaboration of not only computing systems but also user devices (IoTs) and cyber-physical systems located at the edge of the network. Furthermore, we will discuss the probable conflicts and anomalies that arise when heterogeneous systems attempt to communicate and collaborate in a distributed environment and the possible detection and resolution techniques employed by our policy management framework to resolve such conflicts.

Chapter 2

RELATED WORK

Understanding the related work in the area of Big Data, Cloud Computing and Edge Computing and their motivation for the introduction of a new computing paradigm is vital towards conveying the ultimate goal of this research, which is the proposal of a robust policy management framework in a GORE computing environment.

2.1 Related Work on Edge-Computing

Chapter 1 discussed the overall concept of IoT, cloud computing, and the realization of a Big Data problem along with reasons as to why cloud computing will not suffice as an efficient model for real-time communication. This chapter will discuss the related work by leading research organizations so as to establish a platform in understanding our proposed architecture and how it differs from current proposed and implemented systems.

Efforts to realize real-time communication for IoTs have led to leading research groups at IBM and Cisco to introduce two preliminary concepts: Edge Computing and Fog Computing, respectively. Each computing paradigm is oriented to provide IoTs with a platform to leverage resources in real time and obtain services on-demand. However due to their infancy there are a number of components which require further analysis and research.

2.1.1 Cisco — Fog Computing

Cisco research team introduced the concept of Fog Computing and its underlying architecture with concise definitions and use-cases to promote the efficiency and neces-

sity of such a dynamic platform (Bonomi *et al.* (2012)). They focused on two primary use-case scenarios: Smart Transportation Systems and Wind Farm Systems. The Fog architecture consists of three unique layers: (i) IoT verticals: which consists of mutitenant hosted applications utilizing a Fog architecture, (ii) Orchestration Layer API: considered the brains of the architecture, where the analysis, planning, and execution of applications and related data occurs, (iii) Abstraction Layer API: which is designed to hide the heterogeneity of the Fog architecture and provide a uniform programmable interface. Having recently proposed this architecture, the Fog paradigm is still relatively new and fails to take into consideration a number of security criteria which are needed in a heterogeneous system.

Further work on the Fog Computing architecture was pursued by Madsen et al. (Madsen *et al.* (2013)) who evaluated the fog computing platform from a very abstract perspective and have provided certain interesting evaluation criteria that a fog platform should meet. Such criterion was a Machine-to-Machine (M2M) interaction between two connected devices. The fog architecture only takes into account the communication and collaboration between a smart device and its corresponding application, however, by design, an IoT-oriented smart device will have the capability to communicate with its adjacent smart device and share information between them. However, the fog architecture does not provision such capabilities nor does it accommodate for M2M collaboration. A second evaluation conducted by Madsen et al. was the reliability protocols that a fog system should utilize (Madsen *et al.* (2013)). These protocols involve a network-based communication facilitation. However, they did not present any conclusive implementation nor evaluation results of any reliability tests. Cisco has however proposed three different networking protocols to support IoT communication. These protocols include: 6LoWPAN, RPL, and CoAP, all of which focus on a more real-time interaction between IoT based devices.

Although preliminary, the fog framework attempts to realize the need for robustness and real-time communication which previously was not a priority. However, the framework was designed requires more maturity. A more dynamic and fine-grained security model is required for the data that is expected to flow through the fog infrastructure. A key component in such a security model for a fog architecture is a policy management model. The distributive nature of a fog computing environment requires a secure policy management framework to support secure collaboration among smart devices and their corresponding applications. However, the policy management framework described in the current fog computing framework in (Bonomi *et al.* (2014)) is primitive, as the policies described are abstract and specified from a business perspective.

2.1.2 IBM — Edge Computing

In 2003, the IBM research group partnered with Akmai to push Java computing to the edge of the Internet. This new service was called edge computing. Akmai unveiled a new service that would enable users to run IBM's customized WebSphere applications on its network of edge servers. The edge computing architecture consisted of two platforms: The WebSphere software platform and the edge computing platform. The edge computing platform, designed by IBM, consisted of embedded WebSphere application servers which could be accessed via a portal at runtime.

The platform was designed with a data-oriented approach in mind, but did not take into account any distributed system requirements. The focus of the platform revolved around the interaction of applications with the systems and end-users, along with the access to data at the edge of the network. However, there were no clear access control requirements nor provisioning set in place. Davis et al. (Andy Davis (2004)), focused on the deployment of edge computing and the implementation of

distributed applications to be hosted on such an architecture but did not focus on the security criteria that needs to be taken into consideration when a geographically distributed infrastructure with smart communicating devices is created.

Lewis et al. (Lewis *et al.* (2014)), proposed moving cloud computing functionalities to the edge of the network to leverage hosting of data on moving vehicles so as to enable data offloading and dependency on cloud infrastructures. The research conducted was part of the United States Department of Defense funding and focused on the usage of edge-networking for military purposes. However, similar to IBM's approach, the focus of their research was on the provisioning of distributed applications to communicate with their cloudlet services and enabling communication between devices and the edge devices. They failed to consider the access control mechanisms or policy management frameworks that would need to be in place to realize a uniform communication platform over a diverse and mobile distributed infrastructure.

Both research approaches fail to account for two critical functionalities in an edge network: (i) Data distribution and (ii) user access provisioning and management. Without the implementation of these two critical functionalities, an edge network functions the same way as a cloud infrastructure: a centralized intelligence and storage data center.

From the discussions in this chapter, it is clear that there are a number of moving components involved in the collaboration and communication between smart devices and their associated application services. The GORE paradigm, which will further be discussed in Chapter 3 will embody all the previously mentioned requirements. The GORE architecture will focus on a Machine-to-Infrastructure (M2I) model of communication, of which further details will be discussed in subsequent chapters.

2.2 Policy Management and Related Work

Policy analysis and management takes a higher precedence in diverse systems such as fog and edge computing. As iterated before, a GORE paradigm consists of distributed systems with the capability of managing high volumes of user requests. It is thus essential that user privacy and confidentiality be maintained through all communication channels, and unauthorized access to user data is prevented. Ensuring such a high security assurance requires strong enforcement of rules.

In traditional computational models, policies were enforced based on a user's roles and content to which they are authorized to access. Recent researchers (Mansor *et al.* (2012)) have attempted to develop innovative and adaptive policy based approaches, but the levels of interaction taken into consideration are only between a user and a server. They fail to account for multi-system interactions where policies not only have to be implemented from a system to a user, but have to also be implemented from a system to a system based on a user's actions. Additionally, the techniques for policy conflict detection being proposed in recent research (Wu and Liu (2010)) have attempted to resolve policy conflicts dynamically but fail to account for conflicts between different interaction types of a system (Example: conflicts between application level and the operational level of a system) which will be further discussed in Chapter 4.

Traditional policy analysis and management models that we have analyzed (Bertino *et al.* (2009)) were designed for a specific purpose: governing the accessibility and ensuring privacy and security of systems and resources. However, with the introduction of IoT, two additional factors: interoperability and share-ability of systems and resources need to be taken into consideration while designing policy specifications and frameworks. Chapter 4 will discuss detailed specifications and schema upon which

our proposed policy management framework is designed and implemented.

Additionally, there exist numerous approaches related to policy management in distributed computing environments (Teo and Ahn (2007)). There have been significant advances in the area of policy conflict detection and resolution in relation with network policy (Mansor *et al.* (2012), Wu and Liu (2010), Hu *et al.* (2013), Hu *et al.* (2012)). The novel policy conflict and anomaly detection techniques coupled with resolution strategies have been proposed (Hu *et al.* (2012)). However, given the distributive nature of smart devices and the proposed computational architectures, a more dynamic and robust policy management module is required where policy analysis will have the ability to detect policy anomalies and conflicts between different interacting layers of a distributed system and dynamically resolve and provide a decision to the end-user and a system.

The policy management module being discussed in this work is an intricate part of the GORE architecture which will enable multi-tenant applications to be hosted on this platform and enable proper management of resources assigned to individual applications in a heterogeneous system. Such policy management systems will enable a fruitful interplay between multiple diverse ecosystems and end-user smart devices, which is the end target goal of gateway-based computing architecture. Further discussions on the requirements for policy analysis and its integration in a GORE infrastructure are discussed in Chapter 4.

Chapter 3

GATEWAY-BASED COMPUTATIONAL PARADIGM

As discussed earlier there is a need for a robust computational model with the capability to provision services on-demand to clients. These services include but are not limited to networking, storage, and computing. Such a paradigm would need to be situated at the edge of the network where IoTs meet the core cloud data center and would need to be a decentralized infrastructure. Due to such diverse requirements, we name such a paradigm as a *Gateway-oriented Reconfigurable Ecosystem*. Due to its tendency to behave and serve as an entry point for communication requests, GORE is envisioned to be a unique, dynamic, and robust infrastructure for smart device communication.

The architecture being proposed is designed to be innovative and robust, keeping in mind that it is not a replacement of the cloud model but rather an enabler, to support the cloud in performing and provisioning services which it would not otherwise have the capability to do. The model being proposed gives rise to a new era of applications residing at the edge of the network that can proactively communicate with IoT based devices with minimal latency. Furthermore, rather than storing relevant and frequently used data in a centralized location such as a cloud infrastructure, applications will be designed to intelligently utilize essential information while uploading the remaining data for long-term storage into cloud repositories.

A *GORE* infrastructure can be defined as a virtual-driven computing infrastructure with the capability to provision resources including computing, storage, and network at the edge of the network on-demand while serving as an entry point between cyber-physical devices and the cloud computing model.

Similar concepts to GORE have been proposed earlier, which include edge computing and fog computing. Fog Computing (Bonomi *et al.* (2012, 2014)) was designed to seamlessly sit at the edge of the network and provide dynamic, real-time resources to clients. Although our framework embodies certain aspects of the fog architecture (Bonomi *et al.* (2012)), the GORE infrastructure is designed to accommodate components of the fog architecture through fruitful interplay with core components of our proposed architecture. In addition, the GORE infrastructure also introduces unique components and modules to ensure security and collaboration of both data and resources which would otherwise be absent in the fog computing model.

Given the close relationship shared between cloud computing and GORE, it is essential to identify the uniqueness of our model. Our stringent analysis and comparison of current cloud computing models with our proposed GORE architecture, reveals several factors that differentiates our work from other models. Such unique factors are summarized as follows:

- A *tiered organization* involving multiple administrations in a multi-tenant environment.
- A *hierarchical* management and control, supporting interoperable distributed computing environments and interplay with the cloud.
- A distributed and expanded *mobility model* to enable geo-distributed computing capabilities.
- *Geo-distribution* of computational power with extensive focus on service localization.
- *Orchestration* layer involving coordinated control in multi-tier architectural settings.

- *Distributed policy management* frameworks involving multi-tier policy sets and rules.

This chapter will focus exclusively on our proposed architecture and introduce the unique components of the GORE infrastructure which are considered to be differentiating factors when compared with previously introduced computing models.

3.1 GORE Computing Architecture

The proposed GORE infrastructure is a multi-tier architecture involving multiple components. The goal of this architecture is to develop a robust framework which is easily programmable, is flexible, and can host a wide array of applications supporting communication and collaboration over interoperable distributed environments which would otherwise be impractical in a cloud environment.

The architecture proposed in this work as illustrated in Fig 3.1 is a triple-computational service layer structure with a core emphasis on the middle layer. The prime focus of this model is to support a policy-driven service provider for IoT-based devices. Our generic approach toward the design of the GORE architecture enables it to be suitable for a diverse computing environment. Additionally, like the cloud architecture we introduce the concept of multi-tenancy to support flexible usage of the GORE infrastructure.

The concept of multi-tenancy has been utilized in the cloud computing model and has proven to be a successful practice for conservation of resources. Multi-tenancy involves the principle of virtualized resources being utilized by multi-client organizations, all being hosted either on a single server or within the same data center farm. The configurations for multi-tenancy involve the virtual partition of data and services hosted on a single server but on separate virtual instances explicitly utilized and maintained by a single organization. The GORE infrastructure brings

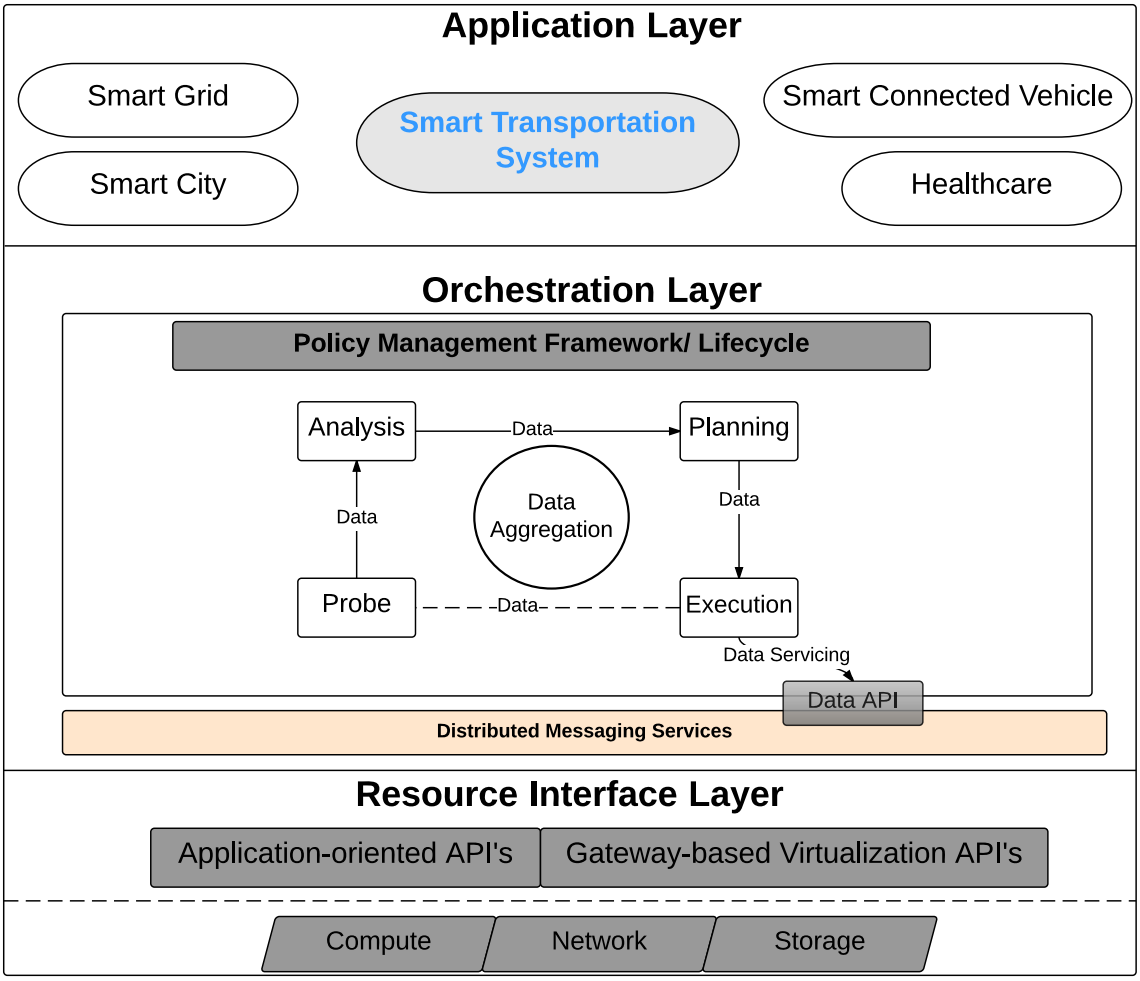

Figure 3.1: GORE Architecture.

this concept to life through the capability for multi-applications hosting on a single node in a GORE environment.

The following are the unique layers of the proposed GORE architecture as shown in Fig 3.1:

1. *Application Layer*: A multi-tenant hosting environment where GORE-based clients can host customized applications. Fig 3.1 illustrates a small portion of a larger pool of applications that have the potential to be hosted on this distributed system. The applications hosted at the edge layer are expected to

deliver real-time data analytics to users on-demand, as well as interact with the orchestration layer of the GORE architecture to determine relevant and irrelevant data based on system and service usage.

2. *Orchestration Layer*: This layer is designed to be the most robust and dynamic component of the overall architecture. The orchestration layer consists of multiple modules, and is responsible for analyzing and provisioning requested services to users on-demand. A major contributing component to this layer is the *Policy Management Framework* module. Additional components include a Distributed Messaging Service for relaying messages between the orchestration layer and its intended recipient, the Data Aggregation module which aggregates all the provided data from IoTs and the cloud data center to provide intelligent data analysis to the policy management module, and finally the Data API module which is a tenant-oriented programmable component, customized to receive and send data in formats recognizable by both IoTs and the orchestration layer components.

3. *Resource Interface Layer*: To provide a flexible user experience for clients, the GORE architecture also includes a third layer: a programmable interface. This interface is designed to allow tenants to develop application-specific services that support the application hosting layer. This ensures a secure communication gateway is established between the application and the orchestration layer to support optimal resource usage and service. This then allows for generic APIs to be developed based on application and tenant requirements which will also be supported by the connecting device.

A careful analysis of the three proposed layers concludes that the orchestration layer is the most crucial component owing, to its robust components, and need for

low latency realization. Therefore, our focus in this work elaborates on this layer which additionally includes the policy management layer.

3.1.1 Orchestration Layer

As discussed previously, the orchestration layer is designed to function with minimal latency. To enable such a realization, the following core modules are introduced:

1. *Policy Management Framework*: This module is responsible for maintaining and governing the functionalities, communication, and distribution of data and resources over heterogeneous environments, while maintaining the highest levels of security through all layers of the GORE model. The policy management framework in particular is designed to not only address access control and policy governance of a GORE model but is extended to regulate the operations and interactions of IoT devices with the system, including the data shared and migrated within the entire GORE infrastructure. Chapter 4 will discuss in-depth architectural details of this module along with concrete analysis techniques, conflict detection, and conflict resolution mechanisms utilized to reduce the risk of both device and system security breaches.

2. *Data Aggregation Module*: This is the intelligent node of the whole framework. The aggregation module can be customized by a tenant, or it can be a third party tool which provides data massaging results based on the data provided by IoTs and user related data pulled from an associated cloud data center. Data aggregation goes through four different generic phases of data massaging. These phases include *probing* the data for relevant information, *analyzing* the probed data to classify, associate, and relate it to historically stored data, *planning* the usage of data, including migration requests, and finally *execution* of the

intended action and usage of data which involves relaying the data to either the messaging service or the policy management framework for further analysis and decision enforcement.

3. *Data API*: This module can be considered as a message translator between the Data Aggregate and the Distributed Messaging Service modules. A simple example to realize the usage of the Data API can be articulated as follows: An IoT sends raw data to the GORE infrastructure system. Before receiving the data into the Data Aggregation module, the Data API parses the data and re-formats it into XML format for systematically parsing and analysis in the orchestration layer. Similarly, once a Data Aggregation module completes the data massaging process, the resultant intelligent data, which is then used to pass a policy decision, needs to be relayed to the IoT. Then the Data API converts the decision into an IoT acceptable format depending on the device receiving the data. A metadata tag can potentially be used to identify the IoT. However this depends on the IoT and tenant communication protocol which requires establishment prior to the initial communication.

4. *Distributed Messaging Services*: This module serves as the point of entry for the orchestration layer. Any request for access to applications, or requests for data access must pass through this messaging service module. This service module is responsible for receiving and sending data messages over a network. IoT networking involves three major protocols, primarily 6LoWPAN, Routing (RPL), and COAP. Although not discussed in detail, the messaging service can relay messages over these networks to the Policy Engine and Data Aggregation module. The messaging service does not make direct connections to the orchestration layer, but rather relays all communication through a Data API which

converts the messages into the desired format for the engine aggregation.

Given the strong justification for elaboration of the orchestration layer, we equally believe that the Resource Interface layer requires close analysis and justification towards its purpose.

3.1.2 Resource Interface Layer

The resource interface layer comprises of two sub-modules: API modules and Resource modules. Each sub-module is interlinked to provide specific functionalities to support the orchestration layer.

1. *Application-oriented API*: These are customized APIs developed by an application owner utilizing a GORE infrastructure to host their application. These APIs are application specific but ultimately are used to send messages/data to the Orchestration Layer via the Distributed Messaging Service. The APIs are developed to accept data from the IoT and relay them to the Orchestration layer.

2. *Gateway-based Virtualization APIs* : These are GORE infrastructure based custom APIs that are developed to be utilized by third party applications and tenants in the GORE system. The APIs can be utilized to support a uniform data aggregation cycle in the orchestration layer thus supporting the policy management framework discussed in Chapter 4. The APIs allow the creation of a pre-defined template that tenants could use to set-up an environment for their applications to interact with the GORE infrastructure modules.

3. *Resource module*: It consists of computing, networking, and storage resources which are provisioned on-demand based on the needs of the orchestration layer.

The resource requests provisioned within the API are then loaded into the orchestration layer for further processing. The resource modules are developed as independent nodes upon which a complete orchestration layer can be supported or as supportive instances, providing resource support for the orchestration layer nodes when resource demand is high.

Based on the activities expected to be performed in a GORE infrastructure, we introduce two resources modules: Gateway Node (GN) and Gateway Instance (GI). Each module is either a cyber-physical or virtual instance which together contributes to the robustness of a GORE infrastructure.

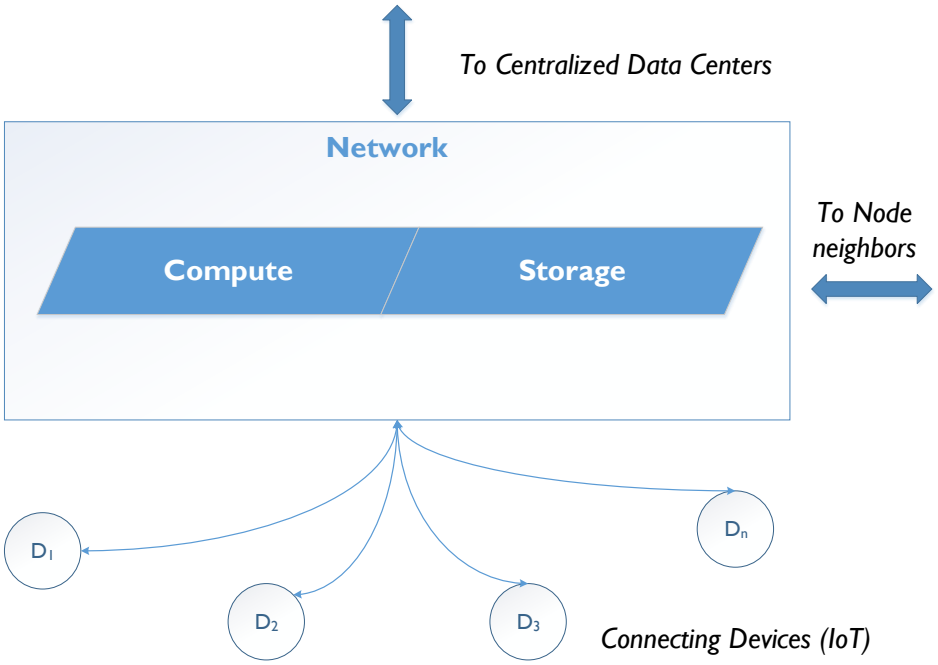

Figure 3.2: Gateway Node.

- Gateway Node: GNs are heterogeneous in nature which enhance their adaptability to the orchestration layer. They have the capability to be

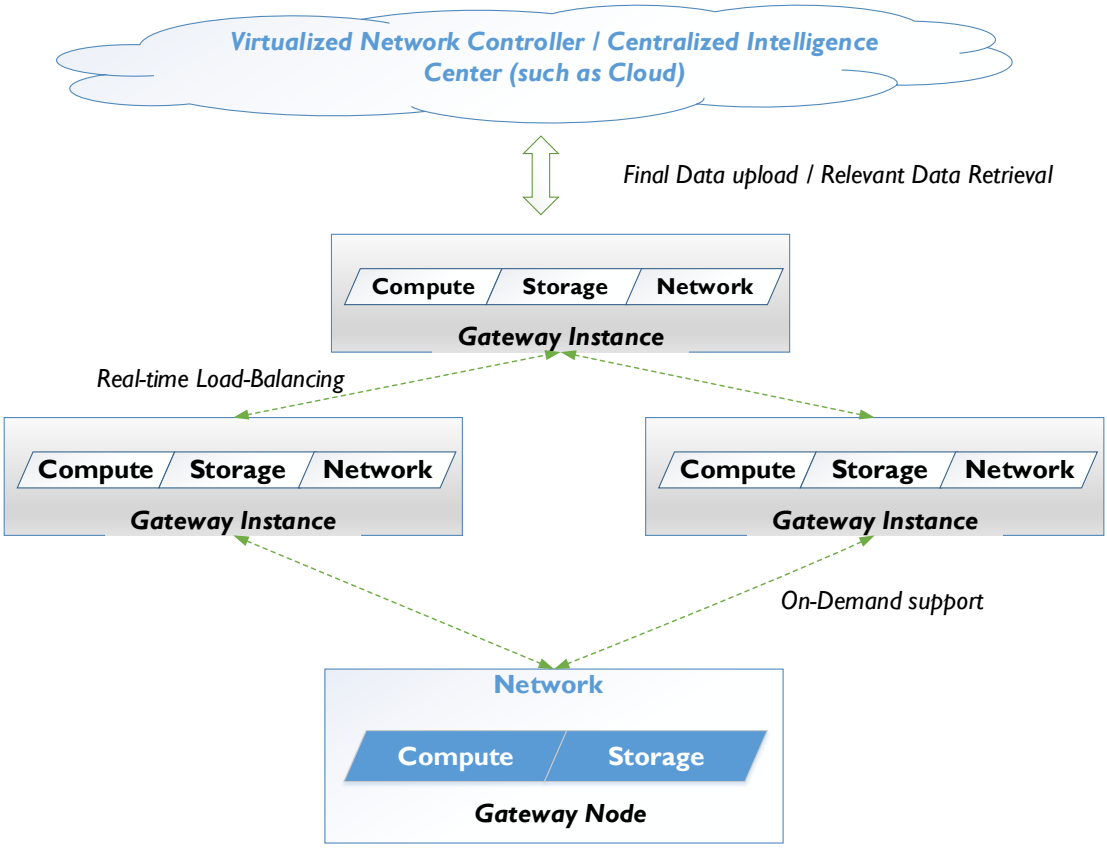

Figure 3.3: Gateway Instance.

deployed in diverse environments and layers such as core, edge, access network, or endpoint environments. They are defined as localized cyber-physical access points from where IoT devices can request services (applications) and resources. The end outlook of GNs is the ability to inter-communicate with adjacent nodes and diverse components in a GORE architecture.

Fig 3.2 illustrates the inner workings of a Gateway Node. The orchestration layer has the capability to reside within the GN and utilize the node for resource and service provisioning. Finally, the performance and capability

of the GN is dependent on its location with respect to an IoT device. Thus, by programming a node as a cyber-physical system, we achieve geo-distribution, localization and real-time data aggregation between a service and an IoT.

- Gateway Instance: GIs are virtualized instances programmed to provide computing, networking, and short term storage service to GNs dynamically as required. They are programmed to behave as support nodes based on resource requirements as illustrated in Fig 3.3. The supportive nature of a GI enables it to provide on-demand resource support to GNs as required. Additionally, GIs are also programmed to distribute resources dynamically based on tenant requirements and application usage. Thus through the creation of a chain of virtual GIs, the realization for a continuous support of resources is maintained in a GORE infrastructure.

Chapter 4

POLICY MANAGEMENT MODULE

Chapter 3 discussed the three unique layers of the GORE architecture with an emphasis on the Orchestration Layer. Given the number of interacting components in the orchestration layer, primarily the policy decision engine, distribution messaging services, data services, and data aggregation services, we believe that policy-based services are a dominant component that requires in-depth analysis and development.

Fog Computing introduced several components in their middle-ware orchestration layer (Bonomi *et al.* (2014)). These components included a foglet software agent, distributed databases, policy-based service orchestration, and scalable messaging bus. The policy-based orchestration framework discussed in their fog paradigm introduced specifications from a very abstract perspective.

Their framework viewed policies from a business point of view but failed to take into account distribution of policies based on stakeholders of their proposed computing model. This implied that the policies did not have a specific schema which is essential for a mature policy management framework. GORE architecture and its subsequent policy management framework provides fine-grained policy orchestration along with a defined schema for generating policy requests and subsequent rules which accomplish the purpose of a secure and robust communication platform for accessing data and applications across heterogeneous GORE environments.

The policy management module involves the presence of a decision making engine that enables the policy-based orchestration framework to enforce policies based on actual interactions in the system. Additionally, our policy module attempts to concentrate on intricate communication modules which will be discussed in this chapter.

These modules make the GORE architecture unique.

4.1 Use-Case Scenarios: Smart Transportation Systems

To better understand and realize the proposed policy management framework, use-case scenarios are utilized to motivate the robustness and interoperability of GORE, along with the moving smart connected components which communicate with a GORE infrastructure.

Our use-case scenarios are based on Smart Transportation Systems (STS). We first demonstrate the role of GORE in supporting STS. STS are intelligent and adaptive systems that accommodate dynamic traffic changes and provide real-time traffic information to travelers by considering potential conflicts and safety issues. A STS environment contains diverse interacting components and each component requests and provides multiple resources and data. Prominent components in a STS environment include Smart Traffic Lights (STL), Connected Vehicles (CV), Emergency Connected Vehicles (ECV), and pedestrian with smart devices. In the use-case scenarios of this work, STLs play an important role as GNs by relaying communication data between other GNs and physical smart devices. STL can be considered as a *System of Systems* of traffic lights in an urban transportation system. Based on the components described above, we propose the following use-case scenarios:

> *Scenario* 1: Bob leaves for work at 7:30 AM and is required to reach his office by 8:00 AM. His office address is stored in the GPS system linked to his CV. When his CV approaches the first STL, it communicates with the STL and receives an optimum route to its destination based on the estimated time of arrival. As he approaches the next STL, based on the current traffic condition factoring in his intended arrival time, the system will update the GPS with the same or alternate route for the requested travel.

Scenario 2: While Bob is on his way to his office, a firetruck, which is categorized as an ECV, travels on the same route to respond to a reported emergency. This ECV will provide the nearest STL with its final destination and by doing so, the corresponding STL will also update Bob's GPS to inform him of an approaching ECV so that he could either prepare to pull over or be provided an alternate route to avoid the ECV.

Scenario 3: A school bus travels to the designated school (eg. Tempe High School) and on its route it makes multiple stops to pick up students, thus affecting the traffic flow. The school bus will communicate its final destination with the first STL, and the STL will then notify all adjoining STLs of the approaching school bus. Since its route is pre-determined, STLs notify all connected vehicles of an approaching school bus and advise an appropriate precaution. The current route is the same as Bob's route and his GPS system will warn him of an approaching school bus.

Scenario 4: Bob exceeds the speed limit and is fast approaching a STL where a pedestrian is about to cross. The STL detects Bob's CV and notifies him via GPS of a probable collision detection. At the same time, the STL notifies the pedestrian of an approaching CV and updates its traffic information to alert adjacent STLs.

The above use-case scenarios help realize the effectiveness of a distributed edge-based computing environment. While computing environments such as grid or cloud depend on virtualization for all its resource allocation, the GORE infrastructure utilizes edge network components such as 4G, LTE, WiFi, and other networking facilities to ensure that uninterrupted communication between interacting components in their system is maintained. Additionally, by sitting at the edge of the network in a geographically

distributed environment, the resources are both physically and virtually distributed, thus ensuring availability of services in regions where GORE architectures have been deployed.

To effectively realize how the GORE architecture can handle these distributed communications, we define a policy schema and associated definitions to design the policy decision engine. This will help understand and measure the complexity of the architecture and its capability to handle dynamic environments such as a Smart Transportation System.

4.2 Policy Management Framework

To account for secure interoperability and accountability of users and services in the GORE architecture, policy definition, collaboration, conflict detection, and conflict resolution are key modules that require mature development and specification. The policy management module is designed to reside in the orchestration layer of either a GN or a GI.

Each smart connected device has the potential to generate policy requests based on available credentials and submit the request to a GN. Each GN has a defined policy definition with sets of rules, to allow device connection and information collaboration. The policies are broadly categorized into the following requirements:

1. Network Management Policies
2. Operation Management Policies
3. Security Management Policies

The classification of policy requirements is necessary due to the heterogeneity of the gateway nodes, making the achievement of collaboration a challenging task. Through the classification of policies based on their different view points as specified above, an

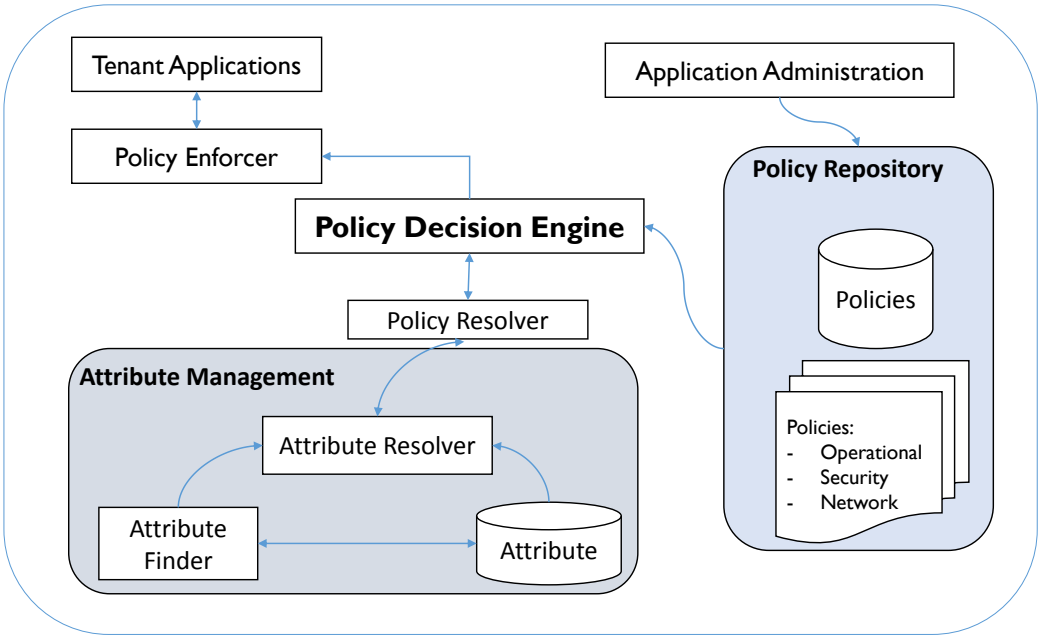

Figure 4.1: Policy Management Framework.

administrator can create policies and rules based on the three defined requirements and store them into a policy repository.

The design of our policy management framework is distributive in nature with each module able to be hosted on separate GI. This allows for the addition of more modules in the future to enhance the framework based on the GORE architecture development.

Fig 4.1 illustrates our proposed policy management framework along with all the intricate components that contribute to its robustness. Currently our framework consists of seven unique modules, and each module performs a specific function that interrelates with other modules. Those modules are summarized as follows:

1. *Tenant Applications*: A dynamic list of uniquely identifiable applications hosted on a GN. This list links to the access points of the listed applications once a

decision of access or deny is made.

2. *Policy Decision Engine*: This module is designed to analyze a request from a client and compare its presented entities against policies stored in the repository. An aggregated decision based on the data provided in the request and its associated policy is made. The decision obtained from the policy decision engine is then relayed to the enforcer module.

3. *Application Administration*: Intuitively, from its name the application administration is the back-end user-interface designed for admin access. Through this interface, an admin can create and upload rules and policies specific to the applications they own within a GORE environment. The created policies are uploaded to the policy repository for access by the policy decision engine. Prior to uploading, the policies are run through a conflict detection and resolution program to parse through, detect and resolve any conflicting policies.

4. *Policy Enforcer*: The gateway that serves as the access point that grants or revokes access to tenant applications based on the policy decision engine outcome. The enforcer is meant to enforce a decision made by the engine and can either reside within the policy management module in the GORE environment or can also reside within the smart connected device.

5. *Policy Resolver*: The policy management framework adopts an attribute oriented security resolution methodology towards identification of an interacting component in a GORE infrastructure. The policy resolver authenticates and identifies users attempting to connect to the GORE infrastructure either through the GN or GI and, based on the set of attributes which they present, validates their identity.

The policy resolver consists of multiple sub-components to verify the identity of the user based on their attributes:

- *Attribute Finder*: This module analyzes the set of attributes presented by a user and queries the attribute database to determine the identity of the user.
- *Attribute Resolver*: Once the identity is determined, it is sent for verification to the attribute resolver to ensure its validity. Following that, the valid identity is sent to the policy resolver.
- *Attribute Database*: A repository of user attributes identifying a users access privileges against a requested resource.

6. Policy Resolver: Once the attribute resolution process is complete, the policy resolver temporarily holds the identity of a user for when the decision engine requires it, for aggregation and evaluation of a request.
7. Policy Repository: A secure repository consisting of rules and policies which are referred to by the policy decision engine when required.

A crucial component in a policy management framework is the policy decision engine. Validation, accessibility, authentication, and authorization are all determined within this component which links to all subsequent components described previously. The next sections will discuss a uniform specification and definition criteria for rules and policies created for applications in a GORE environment.

4.2.1 Policy Definition

We define *policies* as a set of rules against which a client request is evaluated and provisioning of services determined. Each policy consists of a set of rules which are

defined by the following:

1. *Rule id (Rule#)*: A unique identifier assigned to each rule created by an administrator. The rule id enables the identification and detection of conflicting rules.

2. *Subject*: Entities that request resources from a target. A Subject can either belong to virtual components (Gateway Nodes, Gateway Instances, or Cloud data centers) or physical components (smart connected devices such as connected vehicles).

3. *Resource*: Applications providing varied services dependent on the tenants hosting them. The GORE infrastructure allows for multi-tenant applications to be hosted in a single gateway node, thus opening doors to a variety of applications.

4. *Target*: Entities that host applications within their systems. These entities can host multiple applications to which subjects can submit requests for services.

5. *Attributes*: Unique identifiable entities that are associated with a device and user. These entities posses unique properties that are exhibited by both virtual and physical interacting components in a GORE infrastructure. More prominently these properties are exhibited by access control entities such as actors, targets, policies, or any applicable entity, where each entity has the capability to exhibit self-defining characteristics which can be later utilized to identify them (Rubio-Medrano *et al.* (2013)).

   Key attributes in the GORE paradigm are:

   - CUID: Component Unique Identity
   - CType: Component Type

- CLoc: Component Location
- CDestination: Component Destination
- Current_TimeStamp: Date and Time (HH:MM:SS)
- AuthCert: Authentication Certificate (Eg: X.509)

6. Action: Activity to be performed on the resource being requested. Recognized actions in this policy definition include:
   - Access: Ability to view data and resources pertaining to a requested application.
   - Update: Ability to edit data associated with an application owned by that user.
   - Migrate: Ability to move users data between different access point to provide better service provisioning to users on demand.
   - Delete: Ability to remove data belonging to a particular user associated with an application.
7. Effect: Result deduced from the evaluation of a rule in a policy. Permissible effects in a GORE-based policy architecture include:
   - Permit
   - Deny

4.2.2 Policy Specification and Schema

The distributive nature of a GORE paradigm requires a uniform policy specification, that can be utilized across heterogeneous platforms. Policy specifications will formally define how the connecting devices and their hosting environments interact

with the GORE infrastructure. The specifications also assist in defining a structured taxonomy, utilizing common known terminologies in the area of policy analysis while maintaining a unique specification tailored to the GORE paradigm.

To support a multi-tiered architecture while taking into account the different points of policy enforcement, we have classify our policy specifications into three unique categories:

Network Management Policies

These policies focus on maintenance of secure communication channels, network load balancing, and network Quality of Service (QoS) requirements. QoS enables the quantitative measurement of service provisioning. It measures aspects of networking which are essential for its continued, uninterrupted functioning such as error rates, bandwidth, throughput, transmission delay, availability, and more. Network management policies ensure that continued connectivity is maintained by all systems interacting with smart devices in a GORE infrastructure. Additionally, network-oriented policies define networking provisioning constraints imposed on a GORE infrastructure. These include constraints on the communication channels utilized by collaborators and clients in communicating and sharing data over a distributed network.

A simple example would be: constraints on the IoT networking protocols defined in a GORE system. There are three primary protocols designed for use by IoT based devices. These protocols are: 6LoWPAN, Routing (RPL), and CoAP. If an application hosted on a GORE platform is designed to communicate utilizing a 6LoWPAN protocol, while the IoT-based device attempting to communicate with the application is utilizing CoAP to relay data, network policies will be able to specifically identity the communication protocol differences, and based on the application design, re-route the smart device communication via the messaging services in the orchestration layer

and allow for the conversion of the communication protocol to one compatible with the application service.

Similar to the communication protocol example, further networking constraints can be implemented to handle routing of data and to manage the load balancing of the system so as to ensure equal distribution of resource usage. Additional services that a network management policy could define include *firewall policy services* which analyze the rules defined and check for conflicts, anomalies, and redundancy in the defined rules, and finally *network isolation policy services* which focus on prevention of network sharing in multi-tenant environments.

Operation Management Policies

Operational polices are designed to govern the functioning of systems and devices along with their interactions. System functionalities include the type of services a user should have access to, based on his security credentials. Functionalities could also limit the type of services and information shared with users and migrated between different gateway nodes and instances.

Operational policies involve multiple rules that cover a broad range of functionalities:

- *Load Balancing Thresholds*: policies that enable the managing of service loads to ensure thresholds are not breached or exceeded. These thresholds include the balancing of computing, networking, and storage resources within each GI and GN of a GORE infrastructure.

- *Device and Service Instance Configurations*: policies that dictate the pre configuration and management of services and devices. These policies define a uniform template and help in creating an interoperable environment for GORE

infrastructures.

- *Service Attachments*: policies defined by administrators, which when deployed into GNs and GIs, have the capability to recognize and attach valid services to user requests based on their credentials.

Security Management Policies

Policies that are exclusively enforced to dictate the security criteria that must be met to enable secure collaboration and interoperability among distributed GI and GN. These policies, similar to the networking and operational polices are well defined based on the rule definitions discussed in Chapter 4.2.1. The goal and design of security policies are to act as the first line of defense against unauthorized access and usage of GORE resources. Therefore, certain security policies could potentially intersect or overlap with either networking or operational polices, but will however take precedence in the event of such an occurrence.

These policies involve rules covering a wide rage of functionalities. Two prominent policies worth mentioning are:

- *Security Multi-tenant Isolation*: *Multi-tenancy* is the concept of hosting diverse, unrelated tenants in a GORE infrastructure. For example, Google Directional Services, and eHealth (Electronic Health Record) services are two separate services owned by two different companies. However, both service applications are hosted on the same GN. Security policies should be well defined to recognize incoming requests, identify the tenant, and reroute requests to the particular application without sharing confidential data from both the application and the requester.

  Within the application itself security policies will be defined to ensure user cre-

dential verification and validation prior to application access and usage. Finally, the security policy design should ensure policies and their subsequent rules prevent devices and applications from accessing resources to which they have no prior authorization.

- *Security and Privacy*: These policies generically focus on the delivery of data protection across an application or enterprise. They involve the composition of people, processes and technology that contribute to the prevention of unwanted actions that could lead to destruction of data.

  However, in our research, we further explicitly examine and define security and privacy in terms of policies and rules. That is, the concept of security and privacy is a composition of policies and rules that bind a user and his identity to the data that accompanies his credentials while ensuring access to data being requested is provisioned to the user through formal validation and authorization of his credentials.

Our work will primarily focus on the combination of operational and security policies and techniques for detecting and resolving conflicts and redundancy occurrence in a GORE environment consisting of customized policy definitions.

Schema

The proposed policy management framework described in Chapter 4.2 requires a well defined policy definition supported by a mature schema. Given the multi level policy specifications in Chapter 4.2.2, a schema is classified into: Data Schema, and Policy Schema.

1. Data Schema: the first step toward defining a mature schema is the definition of its attributes that define a component or user. The data schema is defined as

a set of attributes associated with both the physical (smart devices) and virtual (GNs and GIs) components of the GORE environment. Listing 4.1 illustrates a small sample definition of a data schema in terms of a Smart Transportation System (STS) environment.

2. Policy Schema: Following the specification of a data schema, we are able to construct a well defined policy schema, utilizing the attributes defined in a data schema. A policy schema can thus be defined as conditions associated with a requested action which, when satisfied, performs the requested transactions. Our policy schema is structured to utilize *XACML* as the policy definition language.

Listing 4.1: Data Schema Specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Document created by: Clinton Dsouza; Gail-Joon Ahn, SEFCOM-ASU -->

<Specification-1 Target="STL1.0" Requester="CV01" Resource="Authentication-Device
    ">
    <Attributes Authentication="X.509" UUID="CV01" GPS-Lat="33.4545" GPS-Long="
        -111.98787" Time="7:30:00pm">CV01
    </Attributes>
</Specification-1>

<Specification-2 Target="FN01" Requester="STL1.0" Resource="Authentication-User">
    <Attribute Security_Token="X.509" UUID="STL1.0" Location="Tempe,AZ" Time="7
        :30:01">STL1.0
    </Attribute>
</Specification-2>

<Specification-3 Target="FN01" Requester="FN02" Resource="Security-Data_Migration
    ">
    <Attribute Security_token="X.509" UUID="CV01" Destination="Mesa,AZ" Origin="
        Tempe,AZ" Target_Subject="FN02" FN02:Security_Token="X.509"
        FN02:Destination="Mesa,AZ">
    </Attribute>
</Specification-3>
```

4.2.3 Policy Analysis

With a clear definitions for policies, rules, and schema in Chapter 4.2.2, we can now discuss the policy analysis techniques developed to support real-time user request

evaluation and service provisioning. Additionally, we also discuss conflict detection and resolution techniques for rules and policies defined by administrators for an application.

While specifications give a clear classification of the types of policies, the schema provides a well defined taxonomy that defines how policies are defined in a GORE infrastructure. To perform fine-grained policy analysis on the requests and their associated policies, and to further evaluate them for conflicts and anomalies, a sample use-case study needs to be established. In Chapter 4.1, we defined four unique use-case scenarios based on a Smart Transportation System and its interaction in a GORE environment.

Utilizing, these scenarios, we define policies and rules based on three sample applications that we envision to be hosted in a GORE infrastructure: Direction Services, User Profile Services, and Health Services. Let us consider use-case scenario 1 and 2: these scenarios revolve around a person's travel from home to work and how the integration with a STS environment would help him efficiently get to his destination with minimal delays. During his travel, his route is intercepted by emergency vehicles responding to an event in the vicinity. Based on the destination of the ECV and the occurrence of the event, the person's route is altered to the next efficient route with minimal delay. Additionally, within the STS environment, Smart Traffic Lights are able to provide the user with real-time event updates including notifications of when an ECV is approaching his vehicle.

From a policy analysis perspective, in these use-case scenarios all three policy specifications: network, operation, and security play a vital role in enabling collaborative communications in a STS environment components. However, we focus our attention on policies related to security and operation. This helps in understanding the role of policies and how they resolve any communication conflicts between mul-

tiple smart connecting components. Table 4.1 illustrates a small set of rules that would be created by an administrator of an application. For example, since we are utilizing a STS environment as an example, the applications are traffic-oriented and subsequently the rules developed are for traffic based applications. The *Resource* defined in these rules are applications hosted in a GORE infrastructure. Each rule specifies a set of conditions which we refer to as *Attributes*, that need to be met so as to gain access to requested resources. The vital entities in these rules are the *Attributes*, *Action*, $Effects$ and *Subject*. Based on a user request each entity of a rule is evaluated against the attributes presented in the request to validate the user and the access privileges he is requesting.

Careful observation of these rules will show conflicts within themselves. For instance: Rule 1 (r_1) and Rule 2 (r_2) conflict with each other, because they overlap in time with a different $Effect$ in place. Additionally, the *Subject* of r_1 and r_2 overlap, thus adding another conflict occurrence between the two rules. Rule 3 (r_3), Rule 4 (r_4), and Rule 5 (r_5) again overlap in time but with the same $Effect$. Because our policy specification follows the defined XACML schema, we could utilize the Policy Combining Algorithms (PCA) defined by XACML itself as a resolution methodology (Rissanen (2013)).

However, the PCA defined by XACML only analyzes two rules at once, and can only analyze one entity at a time. That is, we could direct XACML to perform a Permit Override or a Deny Override which would only take into consideration the effects of each rule but ignore the remaining entities. However, in r_3, r_4, and r_5, we have three rules where the entities are the same but the conditions displaying the attribute *time* are different. This creates a complexity that is challenging for XACML's default algorithms to handle. Hence, there is a need for a more fine grained detection technique that takes into account all the vital entities of a rule, as well

Rule	Subject	Resource	Target	Attributes	Action	Effect
1	CV, ECV	Health Service	STL	{*CLoc:* Mill Avenue & 7th St., Tempe, AZ; *Current_TimeStamp:* 9:59am to 6:00pm}	Access	Deny
2	ECV	Health Service	STL	{*CLoc:* Mill Avenue & 7th St., Tempe, AZ; *Current_TimeStamp:* 1:00am to 11:59pm}	Update	Permit
3	CV	Direction Service	GN	{*CLoc:* McClintock Drive & Apache Blvd., Tempe, AZ; *Current_TimeStamp:* 9:00am to 7:59pm}	Access	Permit
4	ECV, CV	Direction Service	GN	{*CLoc:* McClintock Drive & Apache Blvd., Tempe, AZ; *Current_TimeStamp:* 1:00am to 11:59pm}	Access	Permit
5	ECV, CV	Direction Service	GN	{*CLoc:* McClintock Drive & Apache Blvd., Tempe, AZ; *Current_TimeStamp:* 9:00am to 6:00pm}	Update	Deny

Table 4.1: Security and Operational Rules.

as allows an administrator to specify his resolution preference in the event of rule conflicts.

We utilize a *policy-based segmentation* technique as an approach to closely analyze the rules. However, to understand and realize the effectiveness of this approach a well formed representation of the policies and their rules are required. Binary Decision Diagram (BDD) is a data structure that has been widely used for formal verification and simplification of digital circuits (Hu *et al.* (2013)). We utilize BDDs as a method for simplification of rules for easier verification and integration with our segmentation approach.

Given the policy and rule definitions as illustrated in Table 4.1, it can be parsed to extract vital information stored in the subject, target, resource, action, and effect. This vital information contains attributes that are utilized to make decisions by the policy decision engine. Once these attributes are identified, all XACML rules can be transformed into Boolean expressions. Each expression is composed of Atomic Boolean expressions combined with logical operators AND ($\wedge$) and OR ($\vee$). The atomic Boolean expressions are treated as either equality or range constraints depending on the attribute values and definition. Because we have a structured rule definition, atomic Boolean expressions can easily be derived based on the rules illustrated in Table 4.2.

Atomic Boolean Expressions: Consider rules r_1, r_2, r_3, r_4, and r_5 from table 4.1. In terms of atomic Boolean expressions, r_1 can be represented as follows:

$$(Subject = \text{“}CV\text{”} \vee Subject = \text{“}ECV\text{”}) \wedge (Resource = \text{“}HealthService\text{”}) \wedge$$
$$(CLoc = \text{“}MillAvenue\&7thSt.\text{”}) \wedge (9:59am \leq Current\_TimeStamp \geq 6:00pm) \tag{4.1}$$

Atomic Boolean Expression	**Boolean Variable**
Subject = "CV"	S_1
Subject = "ECV"	S_2
Action = "Access"	A_1
Action = "Update"	A_2
Resource = "Health Service"	R_1
Resource = "User Profile"	R_2
Attribute = "CLoc: Mill Avenue & 7th St., Tempe, AZ"	$Attr_1$
Attribute = "CLoc: McClintock Drive & Apache Blvd, Tempe, AZ"	$Attr_2$
Attribute : 1:00am $\leq$ Current_TimeStamp $\geq$ 9:59am	$Attr_3$
Attribute : 9:59am $\leq$ Current_TimeStamp $\geq$ 6:00pm	$Attr_4$
Attribute : 6:00pm $\leq$ Current_TimeStamp $\geq$ 11:59pm	$Attr_5$

Table 4.2: Atomic Boolean Expressions and Corresponding Boolean Variables for r_1, r_2, r_5, r_6.

Because Boolean expressions are derived from XACML policies and rules, there will be situations where these rules overlap with respect to their attribute values and their ranges. If the attribute values of two different rules are the same they are assigned the same Boolean variable. However, if the attribute values are overlapping ranges, the atomic Boolean expressions needs to be transformed into a new sequence of expressions with disjoint value ranges. Hu et al. utilized a similar approach in their policy conflict detection approach (Hu *et al.* (2013)) and we adopt the similar principles in constructing our Boolean expressions from our defined rules.

Each Boolean expression for a rule is encoded as a Boolean variable as illustrated

in Table 4.2 and as an example Subject = "CV" is encoded as S_1. Utilizing these Boolean variables we can construct a complete Boolean expression in terms of the rules specified in Table 4.1.

Once again consider the rules in Table 4.1 The Boolean expression for rule r_1 is:

$$(S_1 \vee S_2) \wedge (R_1) \wedge (Attr_1 \wedge Attr_4) \tag{4.2}$$

Similarly, the Boolean expression for rule r_2 is:

$$(S_1) \wedge (R_2) \wedge (Attr_1 \wedge (Attr_2 \wedge Attr_3) \tag{4.3}$$

Following the formal introduction and development of Atomic Boolean expressions, the next step would be to define the BDD structures to represent the derived Boolean expressions. BDDs are acyclic graphs which enable the visualization of the previously defined Boolean expressions (Andersen (1997)).

Each non-terminal node in a BDD structure is representative of a Boolean variable with two edges. Each edge has a binary label of a 1 for existent and 0 for non-existent. The terminal node of a BDD represents either a True (T) or False (F) value.

Once BDDs are constructed for the defined rules, performing set operations, such as unions, set differences, and intersections, which are required for our segmentation approach, is easier to achieve. Fig 4.2 illustrates an example BDD structure for rules r_1 and r_2, supported by a set operational function ($\cap$) displaying the resultant BDD structure.

Utilizing the derived atomic Boolean expressions and the resultant BDD structure, a mature policy-based segmentation approach is used. Following the BDD representations illustrated in Fig 4.2, we now discuss the conflict detection and resolution

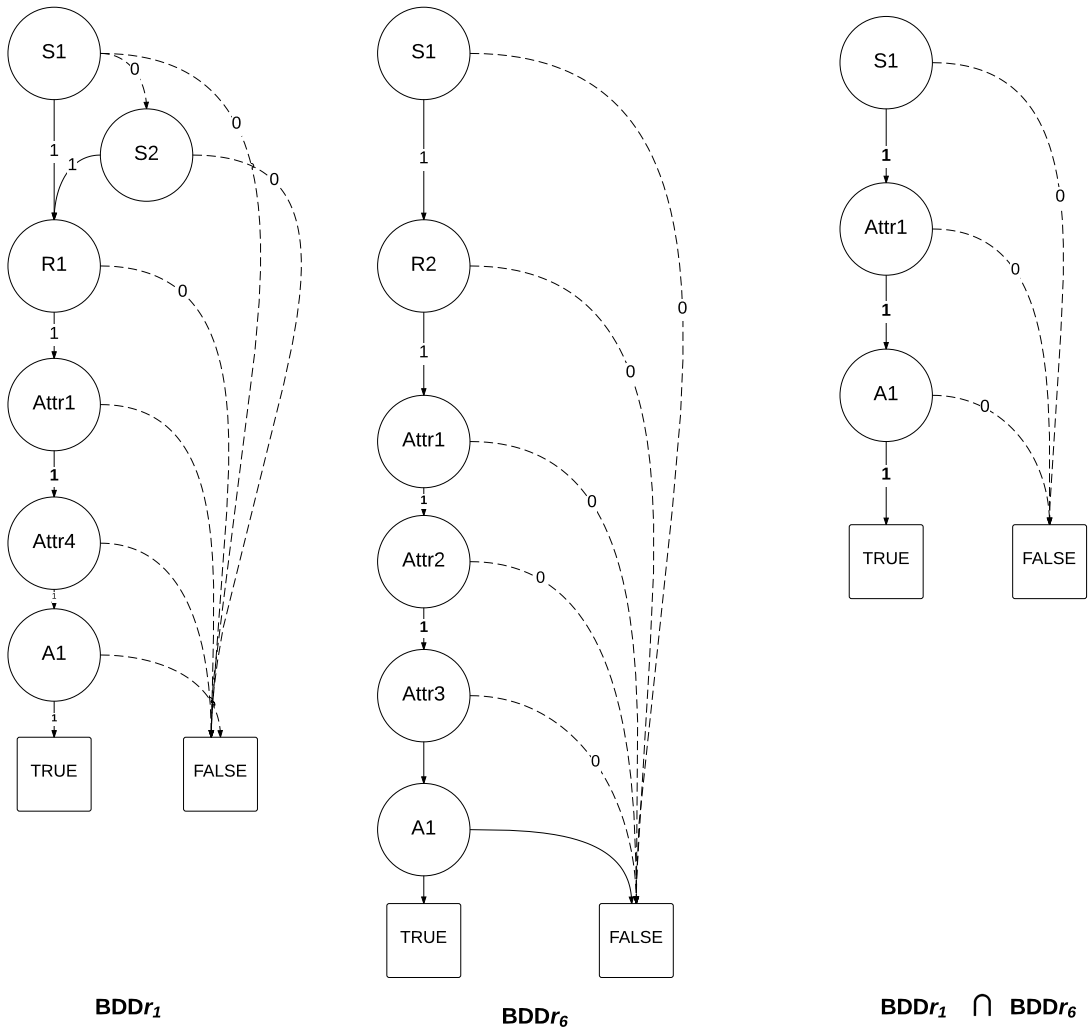

Figure 4.2: BDD Representation of Rules Specification.

techniques that will be integrated into the policy management framework.

Policy Conflict Detection

Our conflict detection mechanism examines conflicts at the policy level, thus allowing for a more fine grained resolution of conflicts. Conflicts at the policy set level are handled by the policy combining algorithms which are defined by the XACML definition (Rissanen (2013)).

For example, consider use-case scenarios 1 and 2 described in Chapter 4.1. Five CVs are approaching a STL and requesting for the same resource: Direction Service, while an ECV is approaching the same STL and requesting the same direction resource to respond to an emergency. In a STS environment, although ECVs are given priority, a STL cannot drop a connection to a CV that is already being serviced. Based on the administrative rules created for the direction services, an ECV is allowed access to Direction Services throughout the whole day.

This creates the following situations:

- The administrator (admin 1), for ease of policy creations, assigned all CVs the same access privileges as an ECV because the service is commonly utilized. This leads to a scenario where an ECV has to wait for critical resources to be delivered.

- A different administrator (admin 2) for the same application assigned shorter time slots to the CVs, so as to ensure that ECVs are provided with essential resources during peak hours. However, without either admin realizing it, the policies specified by admin 2 conflict in time and effect with admin 1.

In Table 4.1, rules r_3, r_4, and r_5 illustrate the above scenario and situations.

To address such situations, our detection algorithm focuses on the policy level to perform fine grained rule analysis. Prior to our discussion of the conflict detection techniques, we first introduce two new concepts that will assist in better understanding of our conflict detection technique.

Authorization Space 1. Let R_x, P_x be a set of rules and policies respectively of a XACML policy x. An Authorization Space for a XACML policy component $c \in R_x \bigcup P_x$ represents a collection of all policy components c to which all access requests Q_c from users can be applicable to.

Attribute Space 1. Consider rules R_x in an Authorization Space of an XACML policy component $c \in R_x \bigcup P_x$. An Attribute Space for a rule R_x represents a collection of unique attributes $Attr_x$ with overlapping, subset, or equivalent values.

Algorithm 1: Identify Disjoint Conflicting Authorization Spaces of Policy P

Input : A policy P with a set of rules (R_x)
Output: A set of disjoing conflict authorization spaces CS_x with associated attribute spaces for P

```
// Partition the entire authorization spaces CS_x for P
// Create a new segment
S.New();
S ← Partition_Policy(P);
// Create a new conflicting segment
CS.New();
// Add all rules associated with a segment
foreach s ∈ S do
    R' ← GetRule(s)
    CS.Append(s);
Partition_Policy(P):
R ← GetRule(P);
foreach r ∈ R do
    s_r ← AuthorizationSpace(r);
    s_ra ← AttributeSpace(s_r);
    S ← Partition(S,s_r);
return S;
Partition(S, s_r):
foreach s ∈ S do
    // s_r is a subset of s
    if s_r ⊂ s then
        S.Append(s \s_r);
        s ← s_r;
        Break;
    else
        // s_r is a superset of s
        if s_r ⊃ s then
            s_r ← s_r \s;
    else
        // s_r is equivalent to s
        if  s_r == s then
            S.Append(s_r);
            Break;
S.Append(s_r);
return S;
```

Conflict Detection at Policy Level: Algorithm 1 illustrates the identification of conflicting authorization spaces of a given policy P and the subsequent identification of attribute spaces. Lines 9-12 partition the rules in a policy into authorization and attribute spaces. While the conflict detection algorithm utilizes the authorization spaces to detect conflicting rules, the resolution algorithm will utilize the attribute spaces to resolve them. The basic functionality of the *AttributeSpace*() function is to extract unique attributes from two or more conflicting rules in a given authorization space. Following the extraction, the attributes are then evaluated to apply a suitable resolution function, which is discussed in the conflict resolution portion of this work.

Algorithm 1 utilizes a function, *Partition*(), from lines 14-28 which accomplishes the task of determining conflicting rules in an authorization space set. This function works through the addition of an authorization space s derived from a rule to an authorization space set S. In order to be categorized as a disjoint conflicting segment, a pair of authorization spaces must meet one of the following relations:

1. *subset* - line 16.
2. *superset* - line 21.
3. *equivalentmatch* - line 24.

Lines 4-6 in Algorithm 1 helps identify conflicting segments. A set of conflicting segments CS has the following properties:

- Pairwise disjoint: All conflicting segments are pairwise disjoint. A pairwise disjoint set is a collection of sets where no two or more elements are shared between the sets.
- Rules in any conflicting segments will have the effect of either *permit* or *deny*.

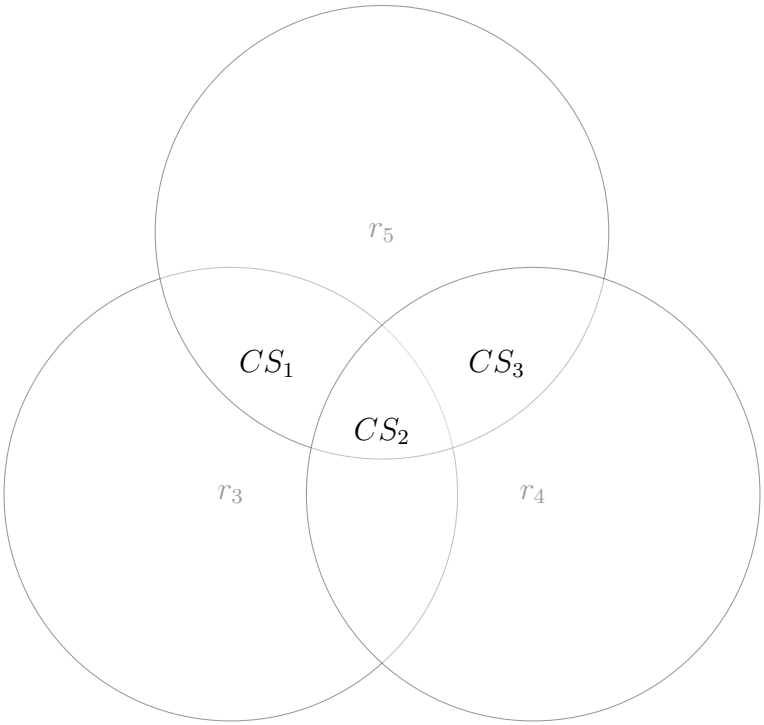

Figure 4.3: Disjoint Segments of Authorization Space for Policy P.

- Requests and Rule relation: Two different requests q and q' that are evaluated by rules within a conflicting segment (cs_i) should be matched by the same set of rules.

We utilized a venn diagram as illustrated in Fig 4.3 to display the conflicting segments and the associated rules within each segment. The venn diagram is one of the many visual methods of displaying conflicting segments. However, even with the visual representation of the rule segmentations, for an administrator, it is still difficult to visually see which rules conflict with each other. Thus a grid representation is utilized to view the rules and their associated conflicting segments.

To enable accurate interpretation of the results obtained from Algorithm 1, we utilized a two dimensional grid representation approach to create an intuitive table.

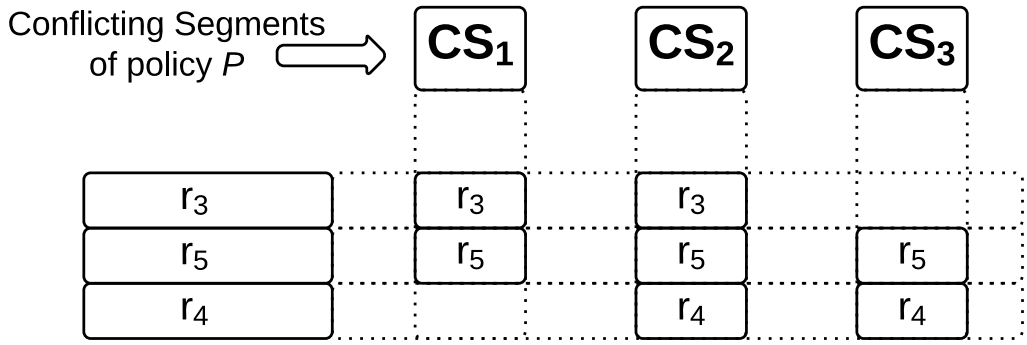

Figure 4.4: Grid Representation of Conflicting Segments CS for Rules r_3, r_4, and r_5.

Although, an XACML rule of a policy P contains multiple fields as illustrated in Table 4.1, for purposes of easy understandability we utilize a two dimensional geometrical representation of disjoint segments rather than a multi-dimensional one.

Fig 4.4 illustrates this grid representation of conflicts in policy P, where rules r_3, r_4, and r_5 represent three rules with conflicting segments. These segments are put into one authorization space. A single rule can be common to multiple conflicting segments. For example r_5 is common to all three segments, while r_3 is common to segments 1 and 2. This grid representation approach helps in better understanding the conflicting rules in a given policy.

Rules r_3, r_4, and r_5 reflect the rules in Table 4.2. The grid representation thus allows us to interpret the conflicts detected in these rules. As discussed earlier, r_5 is a common conflicting rule, and on closer examination will show that because the rule has an effect of *deny* it conflicts with r_3 and r_4. Based on the use case scenarios discussed earlier in this chapter, we observe that the conflicting rules create a situation where an ECV is being denied access to resources during certain time slots which conflicts with rules specified to allow continued access to an ECV. In

addition to conflicting rule effects, there is also a conflict in their time range attribute, which contributes the addition of rules to a conflicting segment and ultimately the visualization of the grid representation.

Following our grid representation of rule conflicts, we discuss the efficient resolution of these conflicts based on both our proposed algorithm and administrator input.

In a GORE architecture, all applications should be designed and managed by a tenant, thus the administrator plays an important role in the design and enforcement of policies for an application. It is therefore important to consider an administrator's resolution preferences, in the event that the created rules result in a conflict.

Policy Conflict Resolution

Resolution strategies can be enforced from two different perspectives: administrative and application. The administrative policies are more static and definitive, and more specifically, focus on determining the operations of an application in a GORE infrastructure. Our policy resolution techniques focus on the administrative perspective of policy conflicts, as administrative policies are the primary reference points in the policy management framework. Because administrators have a greater control over the functionalities of an application, the resultant error on an administrators part could potentially create a higher degree of data and application usage violations.

Our resolution technique requires the following criteria to be met:

- Policies should be specified utilizing the specifications designed in Chapter 4.2.1.
- Each policy should consist of more than one rule targeting resources owned and allocated by a specific tenant in a GORE infrastructure.
- Policies should be compatible with XACML 3.0 request specifications (Rissanen

(2013)). This implies that a user request q should be compatible for evaluation against a supporting policy P.

Once the authorization spaces are determined along with the conflicting disjoint segments, we can begin analyzing the authorization spaces and then further look into the attribute spaces. Each attribute for a rule has a value which, along with the rule effect, will be utilized in resolving conflicted rules.

Algorithm 2: Identify Conflicting Attribute Spaces and Resolve them Utilizing Users Input and Set Operations when Rule Effects are the Same.

Input : An authorization space consisting of rules with same effect
Output: A set of non-conflicting rules for a policy P

```
choice ← UserChoice();
subject ← listofsubjectstochoosefrom;
AttributeSpace (s_r):
foreach s ∈ S do
    if s_r ∈ S then
        s.Attr_i ← GetAttribute(s_r);
    foreach r ∈ s_r do
        if r_i.ID ≠ r_j.ID // where i ≠ j
        then
            if r_i.Effect == r_j.Effect // When the effects of a rule are
               the same
            then
                if r_i.Attr_i == r_j.Attr_j then
                    r_j ← Overwrite(r_i);
                else
                    if r_i.Attr_i ⊃ r_j.Attrj // Attributes of r_i are a
                       superset of r_j
                    then
                        r_j ← Overwrite(r_i);
                else
                    if r_i.Attr_i ⊂ r_j.Attr_j // Attributes of r_i are a
                       superset of r_j
                    then
                        r_i ← Overwrite(r_j);
```

Algorithm 3: Identify Conflicting Attribute Spaces and Resolve them utilizing Users Input and Set Operations when Rule Effects are Different.

Input : An authorization space consisting of rules with different effects and a user input

Output: A set of non-conflicting rules for a policy P

```
1  choice ← UserChoice();
2  subject ← listofsubjectstochoosefrom;
3  AttributeSpace (s_r):
4  foreach s_r ∈ S do
5      if r_i.Effect ≠ r_j.Effect // When the effects of a rule are NOT the
           same
6      then
7          if choice == SubjectPriority(subject) // if the admins choice is a
               "Subject Priority"
8          then
9              if r_i.Subject_i == r_j.Subject_j ∧ user.Attempt()== 2 then
10                 SupersetOverride();
11             else
12                 if r_i == r_j ∧ r_i.Effect ≠ r_j.Effect then
13                     PCA;
14             if r_i.Subject ≠ r_j.Subject∧choice == r_i.Subject then
15                 r_j ←Overwrite(r_i);
16             else
17                 r_i ←Overwrite(r_j);
18     else
19         if choice == SupersetOverride()// if the admins choice is a
               "Superset priority"
20         then
21             if r_i.Attr_i == r_j.Attr_j ∧ user.Attempt()== 2 then
22                 SubjectPriority();
23             else
24                 if r_i == r_j ∧ r_i.Effect ≠ r_j.Effect then
25                     else
26                         PCA;
27         else
28             if r_i.Attr_i ≠ r_j.Attr_j then
29                 if r_i.Attr_i ⊃ r_j.Attr_j then
30                     r_j ←Overwrite(r_i)
31                 else
32                     if r_i.Attr_i ⊂ r_j.Attr_j then
33                         r_i ←Overwrite(r_j)
```

Algorithm 2 and 3 are designed to address resolution of conflicts detected in Algorithm 1. Our resolution strategy primarily utilizes set operations to detect the conflicts and administrative inputs, which are preferences to resolve conflicts. To reduce the comprehensive complexity of the resolution algorithm, we split it into two sections. The first section (Algorithm 2) refers to simple resolutions for when the effects of rules in disjoint segments are the same. That is, rules within a single authorization space have the same effect. The second section (Algorithm 3) focuses on resolution of rules that have conflicting effects. That is, once an authorization space set is determined, we analyze those authorization spaces with conflicting effects to avoid cross-effect policy conflicts from being enforced. Therefore, Algorithm 3 addresses rules with different effects.

Additionally, due to the severity of having two similar rules with completely different effects, our algorithm allows the administrator to provide their preference in the event that such anomalies are discovered. The administrator can provide two preferences: *Subject Priority* — where a specific subject should be given higher priority over others and *Superset Priority* — where rules with larger attribute values are determined to supersede, if it contains all the values of it's conflicting effect-based rules.

Algorithm 3 has certain functions which require clear definitions to understand the fine grained resolution strategy of the algorithm:

- $SubjectPriority()$: This function accesses a list of all possible subjects that an administrator can choose from, as their preference for subject priority.
- $SupersetPriority()$: This is an overriding function, which upon determining the rule to be overridden, removes the redundant rule.
- $Overwrite()$: This is a function that removes an existing rule which is deter-

mined to be redundant or no longer required, either through set operations or via administrative preferences.

- $UserChoice()$: This is a function that accepts a user input in the form of a subject that requires priority in the event of a rule conflict.

- $PCA()$: Represents a Policy Combining Algorithm that is pre-configured by the the administrator when a rule is created. The PCA has four default rule combining algorithms: Permit-Overrride, Deny-Overrride, Permit Unless Deny, Deny Unless Permit (Rissanen (2013)). Our algorithm directs rules to a PCA when the rules are the same but with different effects.

The result of the application of the resolution algorithm on the detected conflicting rules r_3, r_4, and r_5 provided an administrator with a policy consisting of conflict free rules. Because in a STS environment, an ECV is determined to be of high priority due to their role, an administrator would provide his subject priority preference as "ECV". Additionally, because r_4 has a time attribute with a larger time slot in addition to an ECV as one of its subjects, this rule will override rules $r3$ and r_5. This ensures, that the ECV is given a higher priority and unrestricted access to resources it requires.

The implementation of the algorithms described above will accomplish an effective conflict resolution solution for our robust policy management module. Our algorithm not only takes into account the effects of a rule but also considers the attribute values held within each rule, because these values are essential toward the validation of a user's request against an admin-specified policy.

Chapter 5

IMPLEMENTATION AND EVALUATION

Having discussed the architecture of a GORE infrastructure and the different components that contribute toward its efficient functioning, we developed a test-bed that embodied and emulated the functionalities of a GORE architecture. This test-bed is a near-realistic implementation of a Smart Transportation System environment. We define Smart Transportation Systems as advanced systems and applications hosted on either moving or fixed infrastructures that assist in efficient, robust and safe management of traffic systems, making them more coordinated, responsive, and "smarter" to use in a dynamic transportation environment.

A STS is a small component of a bigger plan for a Smart City which consist of mobility networks, smart grids, energy networks, urban analytics, smart buildings and electronic and social networks. A STS within itself has multiple moving components: vehicles, pedestrians, traffic lights, rail-road signals and more. Each component is considered to be "smart". This indicates that these components are either connected or contain devices that can connect to the Internet and upload, share, or migrate data to servers or among themselves.

The US Department of Transportation (DOT) has long been conducting and supporting research in the area of Intelligent Transportation Systems (ITS) which is equivalent to a STS environment. Their research spans a cross-domain area of physical structures and software architectures. Primary research areas of ITS include: safety, mobility, environment, road weather, policy, connected vehicle technology, automated vehicle research, and other supportive resources to assist in efficient functioning of a deployed ITS environment (DOT (2014)). Among the primary research

areas determined by the US DOT we believe *connected vehicles* and *policy* are relevant areas to out work.

Policies can be viewed from different perspectives and so can their implementation and enforcement. While the DOT views policies from an infrastructural and business perspective, we view policies from a more system and user perspective. That is, our policy specifications and definitions in Chapter 4.2 focus on governing the accessibility and operations of users and services in a STS environment.

By definition, a STS requires a robust system that can handle large quantities of user requests while communicating in a distributed environment over heterogeneous networks. Given the number of devices sharing information in a STS environment, a cloud model for service provisioning would be challenging to implement due to the centralized design of cloud computing. However, utilizing GORE as the median between the smart devices and the cloud would serve as an efficient buffer. GORE, being situated at the edge of the network, boosts application distribution and resource share-ability. Additionally, the policy management framework implemented for the orchestration layer of a GORE infrastructure is designed to handle large quantities of user request for validation and provisioning of requested services. This will prove to be an essential component in a STS environment where users are continuously interacting with the transportation infrastructure.

Our STS test-bed design and implementation closely resembles the intentions of an ITS environment envisioned by the US DOT. Additionally, the test-bed enables the evaluation of the implemented policy management framework in an environment closely resembling a GORE architecture. Finally, the test-bed displays the realization of the STS environment requirements through the utilization of GORE infrastructure and its internal policy management framework.

To efficiently evaluate the potential of the test-bed and its performance, we utilize a *metric-based evaluation system*. The primary criteria of this system is *time*. The GORE infrastructure boasts a near real-time provisioning of services as one of the many requirements. The measurement of real-time service provisioning is achieved through the calculation of the time it takes to achieve various functionalities both within the GORE architecture and the supporting systems. Utilizing time as the quantifiable component in the system, we determine two unique metrics that are crucial in evaluating the performance of the test-bed:

- *Conflict Detection and Resolution performance (metric 1)*: This metric is specifically designed to analyze the performance of our proposed conflict detection and resolution algorithm from an administrative perspective. The performance evaluates the time it takes to find all conflicting segments and rules from a set of policies created by an administrator for a specific application and the time taken to resolve them.

- *Policy Resolution and Enforcement performance (metric 2)*: This metric evaluates the time it takes for a server to receive a request, followed with analyzing, evaluating, and generating a response for that request. This metric measures the performance of the decision engine, thus providing valuable insight into the performance of our proposed policy management framework.

- *Attribute Resolution performance (metric 3)*: This metric provides a fine-grained insight into one of the six unique sub-modules of the policy management framework. This metric displays the effectiveness of the policy resolver, more specifically the attribute resolver which identifies a user based on their request against credentials previously stored in the gateway instance database. The resulting time displays the effectiveness of a resolver in validating users, prior to deter-

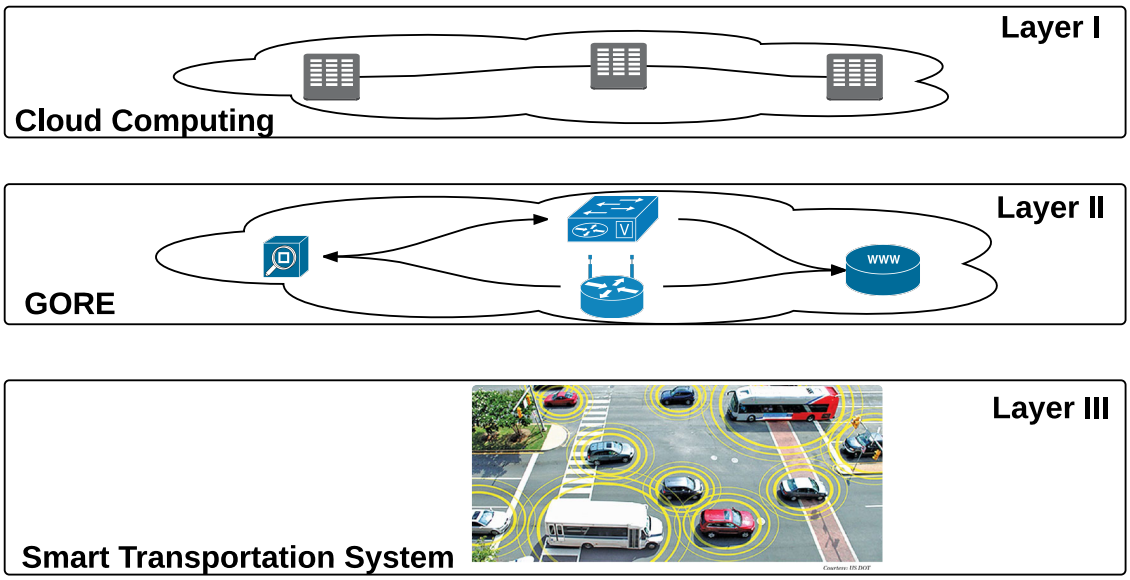

Figure 5.1: Abstract View of the Test-bed.

mining their accessibility to requested services.

Utilizing the above mentioned metrics, the evaluation portion of this Chapter analyzes the policy management framework and the overall system performance of this test-bed while providing vital results for scientific interpretation with regards to the efficiency and usefulness of adding a policy management module to a GORE infrastructure.

5.1 System Design

The STS test-bed was designed utilizing a combination of different technologies, devices and infrastructures. Each component of the test-bed can be classified into layers where each layer is designed to perform a certain functionality. Figure 5.1 illustrates the three layers of our test-bed:

1. *Layer I*: Consists of what is known as the *cloud data center*. We utilize an OpenStack cloud computing infrastructure to create a centralized intelligence

environment that can host instances consisting of STS application data and user uploaded data. The infrastructure also hosts a front-end UI that a user could interact with to study the functionalities of the implemented test bed. Finally, the cloud data center also hosts an administrative interface where administrators can develop policies for their specific GORE-based applications which can then be uploaded to the GORE environment.

2. *Layer II*: Consists of the core components of our work. Primarily, layer II is the GORE infrastructure (including the gateway nodes), which we proposed in Chapter 3, inclusive of the policy management framework proposed in Chapter 4. Current technology limitations, which mainly is the construction of a semi-virtualized networking device with the capability to host applications while providing networking and communication capabilities to these hosted applications, led us to create a semi-realistic GORE infrastructure. To achieve a semi-realistic environment, we employed the use of Raspberry Pi's which are ARM-powered single board computers that provide us with the hardware capability to host applications and also provide networking capabilities either through web interfaces, Bluetooth connectivity, or ZigBee connected devices.

   Finally, we utilized web-based interfaces connected to databases hosted on the OpenStack cloud to achieve both the near-real time requirement and the distributive nature required for a robust GORE infrastructure.

3. *Layer III*: This layer is the actual connected vehicular environment where smart devices communicate with the GORE infrastructure. These devices generate requests for services and utilize the obtained data to make intelligent decisions.

   To achieve this connectivity, we utilized Android-based smart phones loaded with a GPS tracker application that keeps track of the user position as well as

uploads the user position along with unique identifiers to the cloud data center from where the GORE-based components can analyze and perform intelligent computation to assist the user.

Once the complexity of the test-bed is realized, we can further expand on the core components in Layer II of the test-bed.

5.2 Test-bed Implementation

The preliminary test-bed not only focuses on the policy decision enforcement but also on the emulation of a STS based on the use-cases detailed in Chapter 4.1. The test-bed utilizes the Google Maps API for displaying alternative routes as well as real-time route of the user. On the user-end we designed a mobile application that collects user data within a specific time interval. The collected data is stored in a database to be later used for data aggregation. The mobile application also has an added feature that allows a user to enter in the final destination similar to how a GPS device works. Once the final destination is entered, the application sends the device location and final destination to the database, based on the users time interval preference.

When a user provides his final destination, the web application determines a list of alternative routes. As the user approaches a STL, his route changes based on updated traffic information that the STL receives. The STL plays the role of a Policy Enforcer where all necessary policy decisions are routed to the STL for the enforcement on either a CV or a pedestrian. The policy management framework has been also implemented with key elements introduced in Chapter 4. To support policy enforcement capability in our implementation, we utilized ARM-based computers called Raspberry-Pi, which can be emulated to behave as STLs and thus behaves as policy enforcement points. In addition, a generic policy decision engine is built based

on Balana WSO2 opensource policy implementation. [1] The test-bed consists of six

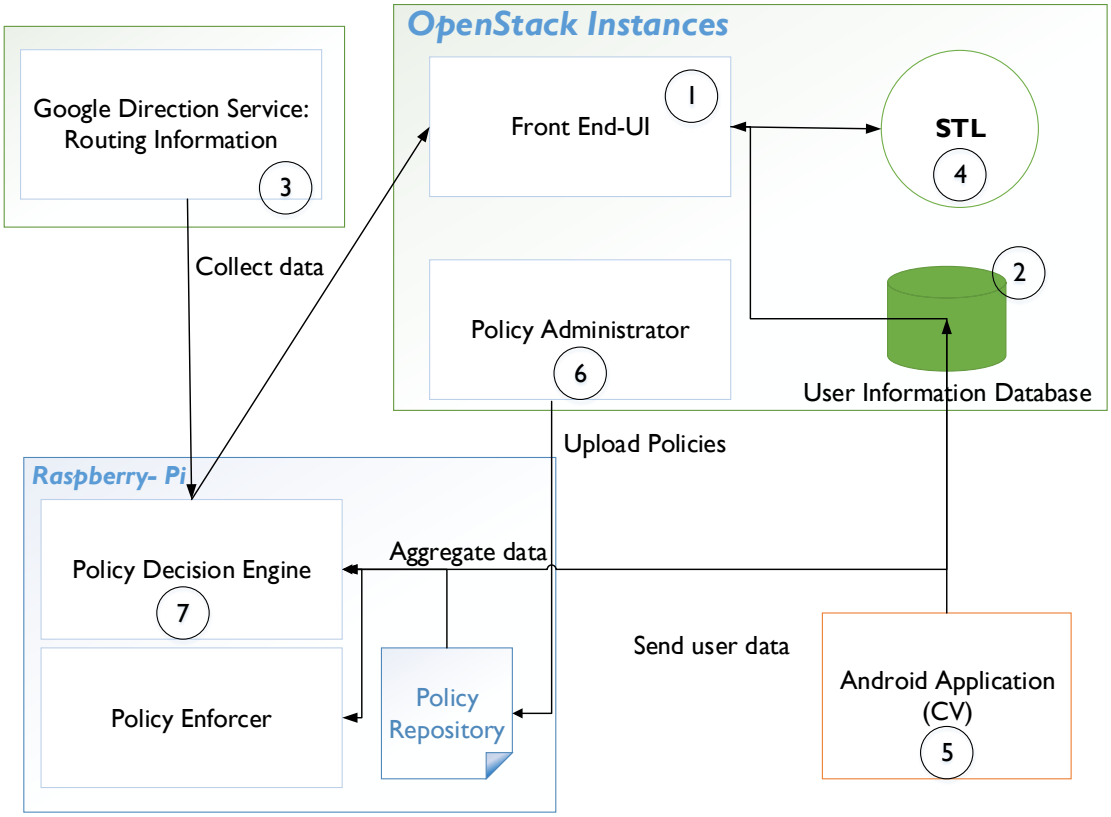

Figure 5.2: Test-bed Architecture.

main components:

1. *User UI*: is the front-end display which a user can use to view his current position and get alerts on policy decisions that are pushed to his device.

2. *Collection Data Container*: is a data object which consists of road information such as traffic delays, road closures, and current road usage by emergency vehicles.

[1]The Balana source code was customized to serve the purpose of an STS environment. Original code can be found here: https://github.com/wso2/balana

3. *Google Directions Services*: are third-party services which are utilized to obtain multiple possible routes based on the data being collected in the collection data container.

4. *STL*: are semi-virtual GNs designed to provide computation, networking and storage capabilities to users and handle requested services. Within the STL resides the policy decision engine wherein all requests for access are evaluated. The Policy Decision engine is the core module of the STL and is vital to the functioning of the test-bed.

5. *Connected Vehicle*: is a smart vehicle completely controlled by the user which establishes secure communication with STLs and generates requests for services.

6. *Policy Administrator*: The back-end UI that an administrator accesses to create policies for a GORE application.

7. *Policy Decision Engine*: The core component in the GN that performs data aggregation and makes a decision.

Also, we utilize and configure the instances in a single OpenStack project to simulate GIs and host the primary web application. The GIs can also behave as supportive services when the GNs run out of memory.

5.3 Policy Decision Engine Design and Implementation

The policy decision engine resides within the GORE infrastructure, as illustrated in Fig 3.1 of Chapter 3. Because a GN is designed to host applications for access by smart connected devices, a Raspberry Pi in our test-bed is considered to be a GN. Therefore, our policy decision engine resides within the Raspberry Pi. However, as with any application, there is an administrative view required to manage the

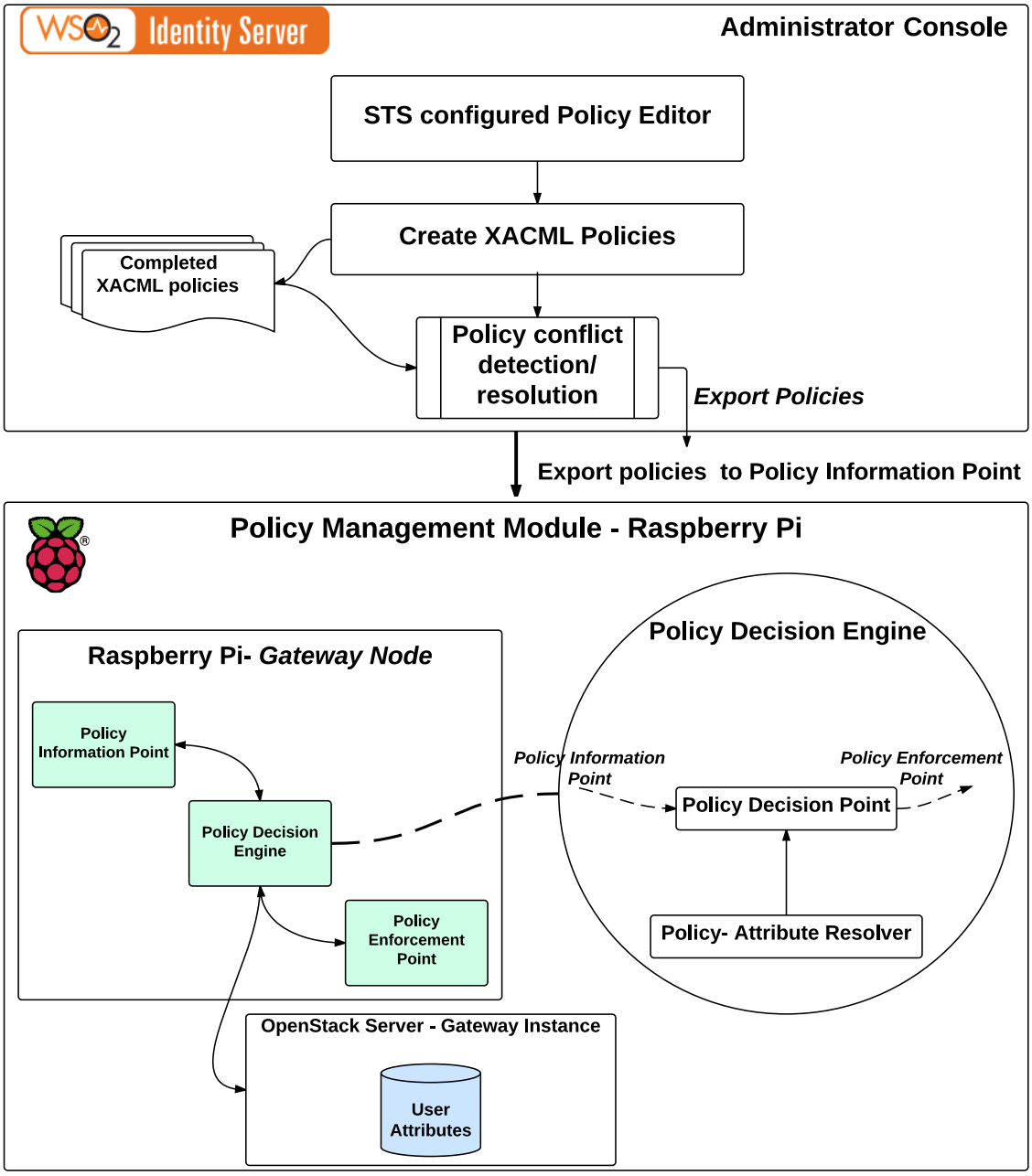

Figure 5.3: Policy Decision Engine Implementation.

application. The administrative view to the policy decision engine is a policy editor through which policies can be created and exported. This policy editor is configured

and implemented utilizing an opensource WSO2 Identity Server [2] . The identity server allows for the configuration of a policy editor to customize it specifically for the development of STS related policies.

Fig 5.3 illustrates the completed connection between the administrative console and the policy management module hosted on the Raspberry Pi.

- *Administrator Console*: As discussed earlier, the administrator console enables the administrator to enter policies related to a STS application hosted in a GORE infrastructure and export it directly to the policy management module on a GN. Additionally, the administrator console also consists of a policy conflict and resolution module that analyzes the administrator-created XACML policies for conflicts and resolves them. In addition to automated conflict resolution, the module also allows an administrator to provide his preference as to the type of resolution they would prefer to resolve conflicts. Once the conflicts are resolved the policies can be exported to the Raspberry Pi.

- *Policy Management Module*: This module is hosted on the Raspberry Pi as part of the GORE architecture. The policy management framework discussed in Chapter 4 is the core intelligence of the GORE architecture. All policies exported by the administrator are stores in the Policy Information Point (policy repository), which the decision engine can access whenever a request is made. A policy request is generated from any smart connected device in the STS environment. In this case, the front-end UI sends requests for access to an application hosted on the GN. In our test-bed the "Direction Services" is an application that routes a connected vehicle to its fastest route based on traffic conditions.

[2]A product of WSO2, an open source platform: http://wso2.com/products/identity-server/

The decision engine evaluates a user request against the policies created by the administrator. Based on the credentials (attributes) presented in the request, the decision engine utilizes a policy-attribute resolver to determine the validity of the request. Once the validity is determined, a decision of *permit* or *deny* is issued, which is then enforced by the Raspberry Pi or the connected vehicle.

5.4 Policy Engine Evaluations

Evaluation of the policy decision engine is a complex process involving the combination of multiple components in the test-bed. The evaluation process is assessed from two perspective:

- Administrative perspective: which involves assessing the performance of the policy conflict and resolution module hosted on the administrative console.
- System perspective: which involves assessing the performance of the policy decision engine hosted on the Raspberry Pi as part of the GORE infrastructure.

5.4.1 Administrative-Level Policy Conflict Detection and Resolution Evaluation: Metric 1

The design of the administrative console is based on the WSO2 Identity server. A GI is created to host the administrator console. The GI for this test-bed is a virtual Windows 7 instance. Once the identity server is created, a user can log-in and create application- specific XACML rules and policies. Once the policies are created they are exported to a folder and then analyzed for consistency.

The analysis of these rules is in accordance with the algorithms defined in Chapter 4 (Algorithms 1, 2, 3). A custom policy conflict detection and resolution Java applet was developed which contains the algorithms developed for the policy man-

Assessment	No. of Rules	Time (ms)
1	165	48
2	195	57
3	225	128
4	255	135

Table 5.1: Metric 1: Policy Conflict Detection and Resolution Time vs Number of Rules.

agement module. Algorithm 1 analyzes the rules in all policies and determines the conflicting segments. Algorithm 2 and 3 analyzes all the specifications or entities in a conflicting rule and resolves the conflicts through removal of the conflicting or redundant rule.

The design of the Java application is such that, once the folder containing all the exported policies and rules are connected to the application, the conflict detection and resolution occurs simultaneously. Table 5.1 illustrates four unique evaluations of the policy conflict detection and resolution algorithms. Utilizing the specifications in metric 3, we evaluate the time the program and its implemented algorithms take to analyze the rules and detect all possible conflicts while resolving them.

It is also important to note that with every assessment increment, the complexity of the rules increase. The complexity was introduced in forms of rule redundancy, effect, attribute, and subject conflicts. This was done to evaluate the stability and efficiency of our conflict detection and resolution algorithms. The overall performance of the program was deduced to be stable with time increasing as the number of rules did. Even with the increase in time, the maximum being 135 milliseconds (ms),

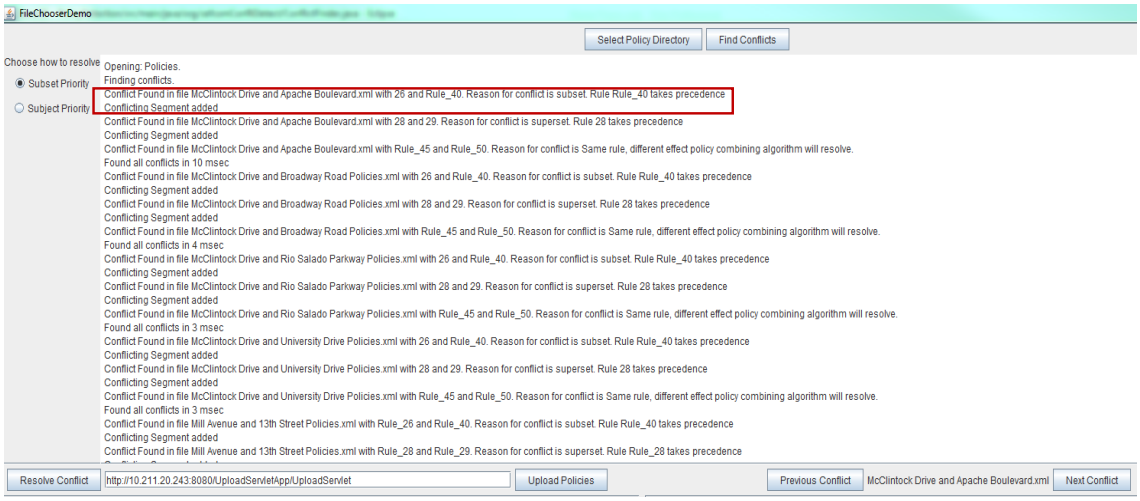

Figure 5.4: Conflicting Rules Detection.

the program was still able to efficiently detect and resolve conflicts in under 200 ms, which compared to the policy decision engine, is still on-par with the overall evaluations conducted in the test-bed.

Given the complexity that an administrator can introduce in an application's policies, our algorithm takes into consideration their preference for resolution as well. We provide the administrator with two choices: Subset Priority or Subject Priority. These two resolutions are taken as inputs in our algorithm while resolving the conflicts. We introduced 12 policies with 15 rules each for our assessment. Fig 5.4 illustrates the obtained conflicting segments and the rule names. The red box illustrates actual conflicting rules on our test-bed. We selected "Subset Priority" as an administrator preference. In this case Rules 26 and 40 are conflicting due to a subset clash. This means that two or more values in a single entity column of the rules are conflicting with each other.

Our console allows the administrator to view the two rules so that he can determine what the conflicting values are. Fig 5.5 illustrates the console view of two conflicting rules. The red boxes indicate the conflicting values. As can be observed, there

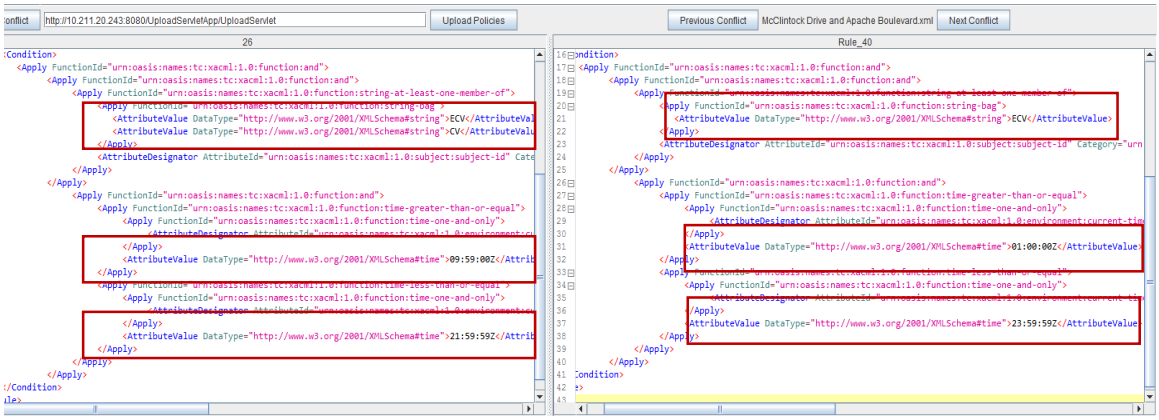

Figure 5.5: Detailed View of Rule Conflicts.

are two unique columns in a rule that are conflicting. In this case, the *"Subject"* and *"Attribute"* are the two rule entities that have conflicting values. Because the administrator has provided his preference of a subset priority, our algorithm will analyze the attribute conditions of each rule and determine which rule covers a broader condition. If the administrator had provided his resolution preference as "Subject Priority", our algorithm would give preference to the rule containing one subject with "EV" as its value. Utilizing this custom developed console, an administrator can ensure that the rules created for an application in a GORE infrastructure are conflict free. Once the conflicts are resolved, the console allows the administrator to push the rules directly into the GN to be stored in the policy repository of the policy management module.

5.4.2 System-Level Policy Enforcement Evaluation: Metrics 2 & 3

Table 5.2 is an evaluation of the average policy enforcement time in a smart transportation system against the number of vehicles interacting in the system. This evaluation covers the first metric discussed earlier (metric 2). Successful evaluation is based on the completion of the following workflow connectivity: A CV approaches a

Assessment#	Average Enforcement Time (ms)	Number of Vehicles
1	115	1
2	419.5	2
3	868.5	3
4	1050.5	4

Table 5.2: Metric 2: Policy Enforcement Time vs Number of Vehicles.

STL and requests for a particular service (direction service in this use case scenario). The STL takes the user request and sends it to the policy decision engine hosted in the GN. The STL could function as a standalone GN or a node, in a cluster of STLs communicating with a separate network-enabled device functioning as a GN. Once the request is evaluated by the policy decision engine and a decision made, it is enforced by the STL onto the connecting vehicle. The decision could range from a simple comment such as a *permit* or *deny*, or could be more complex such as warnings or instructions informing the connected vehicles about changes in their commute times or other emergencies in their surroundings.

Our evaluation shows that the increase in the number of connected vehicles does effect the enforcement time, however, the difference does not significantly reduce the performance of the engine. We observe a minimalistic 20% increase in enforcement time when a maximum of 4 vehicles are communicating with a single STL in the system up from 3.

The design of our system allows us to perform fine-grained evaluation of even the smallest component of our policy management framework. For example, a key sub-module in the policy management framework in the policy resolver, which is an attribute resolver that validates user credentials to determine access to the services

Assessment	Average Attribute resolution Time (ms)	Number of Vehicles
1	3	1
2	7	2
3	12	3
4	16	4

Table 5.3: Metric 3: Attribute Resolution vs Number of Vehicles.

being requested. Due to the distributive nature of our policy management framework, we have the capability of evaluating the attribute resolver (metric 3) from the time a request containing a users credentials is received to when they are verified.

The average time the attribute resolution module takes to resolve a request and identify if the user making the request is validated is illustrated in Table 5.3. As can be observed, the average time for resolution increases with the increase in load on the GN, however, the increase is in milliseconds. When a maximum number of four vehicle are connected to the same STL requesting for services, there is a 33.3% increase in resolution time, however considering the time increments is in milliseconds, there is no significant performance degradation.

5.5 Test-bed Performance and Evaluation

The complete test-bed evaluation involves the performance calculation of a combination of multiple components across a heterogeneous infrastructure and systems. The test-bed evaluation begins from when an administrator uploads a set of application policies into the policy management module. What follows is the workflow criteria established for the overall complete evaluation of our test-bed.

Once policies for a particular application are uploaded, they are stored into a

Assessment#	Average Decision time (ms)	Number of Vehicles
1	589	1
2	1887.5	2
3	3419	3
4	4149.25	4

Table 5.4: Total Policy Decision Time vs Number of Vehicles

policy repository. The policy repository contains all policies that an administrator creates and uploads. When a user connects to an STL, his smart device generates a request in XACML and sends it to the STL. In our test-bed, the STL is a GN hosting the policy decision module. Once a request is received, the policy decision engine first calls the policy resolver which extracts the attributes of the user from the request and compares it with the values stored in a remote database in a GI. Once verified, the user's request is evaluated against policies stored in the policy repository based on the request and service type. Once the policy decision engine evaluates the request, a decision is generated which is then sent to the policy enforcer for enforcement. The policy enforcer pushes the final decision to the user. Based on the decision pushed, the user makes the necessary changes thus completing a successful secure communication in the GORE infrastructure.

Upon the establishment of this workflow, the performance is calculated as the total decision time from when a request is sent to when a decision is made in the GN versus the number of vehicles in the system.

The overall test-bed performance is condensed and illustrated in Table 5.4. As expected, with the increase in the number of vehicles the average decision time also increases. This is attributed to the number of requests begin generated and sent to

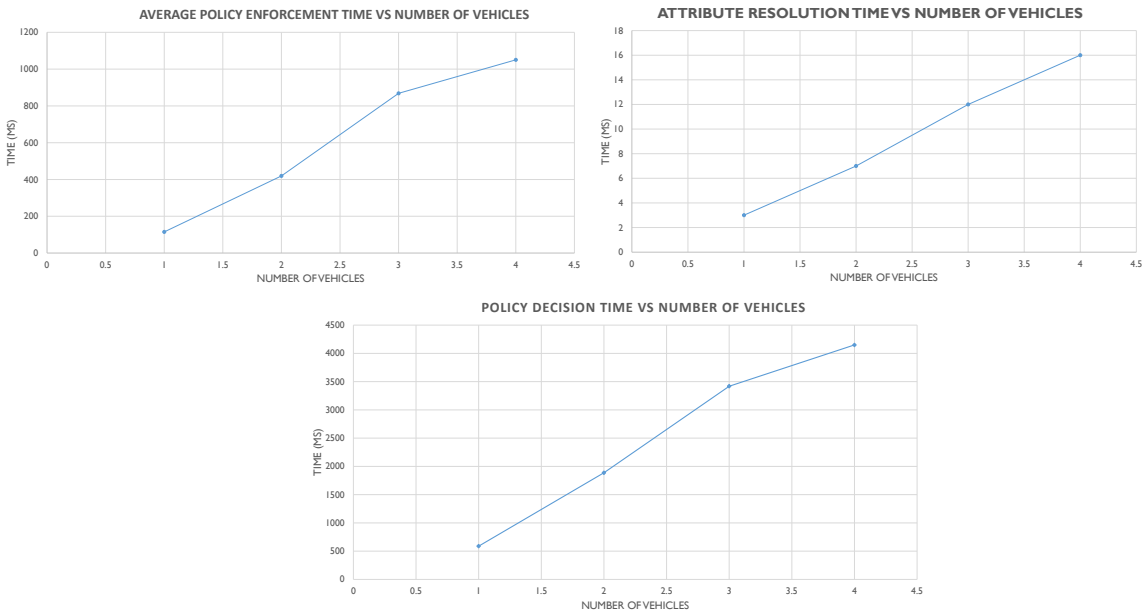

Figure 5.6: Visual Performance Evaluation of Test-bed.

a single STL. However, the final assessment with four vehicles shows a 20% increase from three vehicles, which is not a significant impact to performance. However the total decision time reaches approximately 4s, which indicates that the test-bed is achieving a near-real time communication between the smart traffic lights and the connected vehicles but is still not achieving a complete real-time workflow.

Finally, to better understand the performance of the test-bed under varying system and user loads we utilize intuitive charts as illustrated in Fig 5.6. Through the varying evaluations conducted on the implemented test-bed, we are able to prove that the GORE architecture proposed in this work is capable of providing a robust and efficient infrastructure for secure collaboration and communication of data between IoT devices and its connecting applications. Additionally, the policy management module proposed in this work does enables the GORE infrastructure to function at an optimal level while maintaining a certain level of security and ensuring that user

credentials are validated prior to servicing their requests. With the support of the designed policy management framework, our test-bed is capable of accomplishing these objectives while maintaining a high performance rate and ensuring users are provisioned with validated services within seconds.

Chapter 6

DISCUSSION AND FUTURE WORK

The GORE model like Fog computing (Bonomi *et al.* (2012)) is still a very preliminary model that requires more maturity and supportive infrastructure to enable the realization of an edge-based infrastructure. In January 2014, CISCO announced the release of a new range of networking infrastructure which they called *Cisco IOx*. These were programmable routers with the capability of supporting an operating system within the router itself, thus enabling the virtualization of resources and enabling the hosting of applications (Cisco (2014)). However, the final product has not been released as of this writing. The specifications for an IOx device closely resemble our specifications for a GN, thus leading us to believe that the realization of a GORE infrastructure is a viable option that will be pursued by the industry in the future. Based on the contributions in this work, there are two areas of research in which future work can be pursued: the GORE architecture model and the policy management framework.

6.1 Discussion

Prior to discussing the potential future work that can be pursued in the area of GORE and policy management, we will first discuss the current limitations of our approach and points of consideration that need to be taken into account when pursuing future research in the area of edge networking paradigms.

The current focus of this work has been on proposing a policy management framework that will enable connected devices to coherently communicate and request for services from applications being hosted in a GORE infrastructure. However, our approach does not take into account the authentication and authorization of users to

avoid identity spoofing in the GORE infrastructure. Although our attribute resolution validates a request, there are no security checks that enable the detection of malicious attacks such as, attempts at user impersonation to access sensitive data. Our approach assumes that a user would not attempt to impersonate another user, and the security measures that we implemented ensured that access to resources that are being requested is first validated prior to assignment. This ensured that the communication and sharing of data is completed meaningfully through the evaluation of related policies in the decision engine.

In addition to these security checks, we also need to address the issue of low latency. As observed from our evaluation, we were able to achieve a near-real time communication. However, this can be further enhanced through the utilization of advanced communication layers such as SPDY or the HTTP 2.0 protocol. SPDY has proven to be an effective communication protocol in the transport layer of the OSI model and can potentially be utilized as a method to wrap communication requests and responses between vital components in the policy management framework. The design of this protocol was primarily to enable low-latency transport of data over the web (Belshe and Peon (2012)).

Finally, an important area for future exploration that needs to be looked into from a STS perspective is the hardening of security on STLs. STLs are basically GNs and are the primary point of contact for connected devices to request for provisioning of services. This makes them an attractive target of attack from a malicious user's perspectives. Therefore, an effective threat mitigation model needs to be developed to ensure that in the event of an attack on a STL, protocols are put in place, to ensure continuity of a STS environment is not compromised.

6.2 GORE Computing Model Enhancements

The GORE architecture is designed to contain independent modules that can be loaded into the system to behave as part of the system. In our GORE architecture, we defined a policy management module which is considered to be a small component for ensuring secure communication and collaboration. However, our work does not cover additional security modules that need to be developed to further secure the architecture. Thus, future work in this area could involve the development of a security service module for a GORE environment. Each GN and GI will host a complete or a fraction of a security module, depending on its configurations and requirements of the infrastructure.

A security service module could potentially consist of services such as: Policy Management framework and life-cycle management, Secure communication encryption, Intrusion Detection, Identity Management, and User entitlement services. Each of these services, although specified to be within a single security module, have the capability of being hosted as independent modules thus making the security service module a flexible future development for the GORE architecture.

Another interesting and challenging area of future work in the GORE architecture is a health detection system. Currently, the GORE architecture is designed to be distributive in nature based on a geographical location. However, the current architecture does not contain services or modules to monitor the health of a GN or GI in a distributive environment. Our architecture can, however, monitor resource usage and is capable of handling resource loads (load balancing) based on the policy management module and its specified networking and operational policies.

The health monitoring service would need to dynamically monitor GNs and GIs in its vicinity and have the capability to take over the functioning of a GN or GI

or spawn a new GN or GI in the event of a node being compromised or temporarily being put out of service due to system failure or a cyber-attack. This will enable the GORE architecture to function as a self-sustaining architecture with the capability of supporting and spawning nodes and instances in the event of an attack on systems in a certain geographical area.

6.3 Policy Management Framework Enhancements

The policy management framework discussed in Chapter 4 provides well defined specifications for the policies and rules created in a GORE environment. Furthermore the rules and policies are classified based on their types into: Operation, Security and Networking. This provides us with a multi-layered policy structure for when they are stored in the Policy Repository.

However, in a GORE architecture, the entire infrastructure consists of three distinct layers: the core, which is primarily the cloud data centers, the edge, which is the GORE environment; and the physical layer, consisting of IoT devices. Each one of these layers has specific policies that require enforcement. Our current policy approach takes into consideration all three layers but combines their policies into three policy requirements: Operation, Security and Networking. While these policies are relatively static in nature and are analyzed to a fine-grain level by our policy framework, we believe future work in this area would be the further classification of the rules and policies. That is, utilizing a three-pronged approach of classifying the policies into virtual, tenant, and client, where the virtual policies focus on rules determined to be issued for the centralized cloud data centers, the tenant policies are specified for a specific application hosted on the GORE architecture, and, finally, the client rules are those concerned with IoT device communication with either the cloud or GORE infrastructure. Utilizing this approach, a multi-dimensional policy struc-

ture could be created through which a more robust conflict detection and resolution algorithm can be produced.

Finally, a realistic future work that would enhance the current policy management framework would be the inclusion of a dynamic policy generation module that would continuously evaluate the GORE infrastructure and its resources, usage, and functionalities, and generate policies to better manage the GORE environment and its communication infrastructure. David Putzolu (Putzolu (2003)) patented a unique technique for dynamic policy generation for network management systems utilizing instruction generation based on network condition evaluation. These instruction are policies which are then installed on the network devices within the network management system. Because this concept only focuses on policy-based network management systems, future work would involve the development of dynamic policy generation for other previously classified policy types in the GORE architecture.

Chapter 7

CONCLUSION

This work proposed two new concepts that are intended to enable the Internet of Things based devices to communicate more securely, efficiently, and robustly in a dynamic environment. The *first concept* utilized the notion of edge-networking. We proposed a computing architecture that would reside below the cloud but above the IoT devices. Our architecture boasts characteristics such as homogeneity in a distributed environment, on-demand access, low latency servicing, and geographical localization of services. Additionally, the architecture is proposed not as a replacement but as a support infrastructure to enable the cloud computing model to provide services which it would otherwise not be capable of provisioning. We call this architecture: Gateway-Oriented Reconfigurable Ecosystems.

Within the GORE architecture, we identified various security requirements that would need to be accomplished to ensure secure communication and collaboration. From these security criteria, we identified policy management system as a core security criterion. By taking a policy-oriented approach, we enabled the GORE infrastructure to perform real-time service provisioning based on user requests. Following this identification, our *second concept* involved the proposal of a *Policy Management* module that is placed within the GORE architecture, more specifically within the orchestration layer. We identified the different stakeholder or beneficiaries of the GORE infrastructure, primarily the IoT users and application owners. Furthermore, we developed a mature policy structure for policy and rule specifications, thus designing a relatively mature policy management framework for the GORE infrastructure.

Finally, we proved the effectiveness of our proposed GORE infrastructure and its

supporting policy management module through the development of a test-bed. This test-bed was developed utilizing Smart Transportation Systems as the base foundation and displayed the effectiveness of our GORE infrastructure and its policy management module in realizing a near real-time and low latency communication environment.

7.1 Contributions

The primary contributions of this work are: (i) creation of an edge-networking based computational paradigm called GORE and (ii) the integration of a policy management module within the GORE architecture to validate, authenticate, and authorize user requests for applications hosted with a single node in a GORE environment.

7.1.1 GORE Computing Contributions

Proposed as a potential solution to real-time, on-demand access to robust services, the GORE infrastructure provides smart devices with the capability to request and receive services in low-latency networks in a distributed environment. Utilizing the power of cloud computing coupled with GORE, interoperable communication between distributed services and infrastructures can be established with minimal latency and loss of data. The GORE environment consists of three layers. The primary contribution of this work resides in the orchestration layer. The orchestration layer is designed to function as the core component of the GORE architecture, ensuring every request and communication received by a node in the GORE infrastructure is validated, analyzed and aggregated to produce the most efficient and reliable result.

The uniqueness of the GORE infrastructure falls within this "Orchestration Layer". This layer provides core functionalities and maturity which otherwise is absent in other similarly proposed edge-based computing models. The orchestration layer by design, analyzes all incoming communication and collaboration requests with the as-

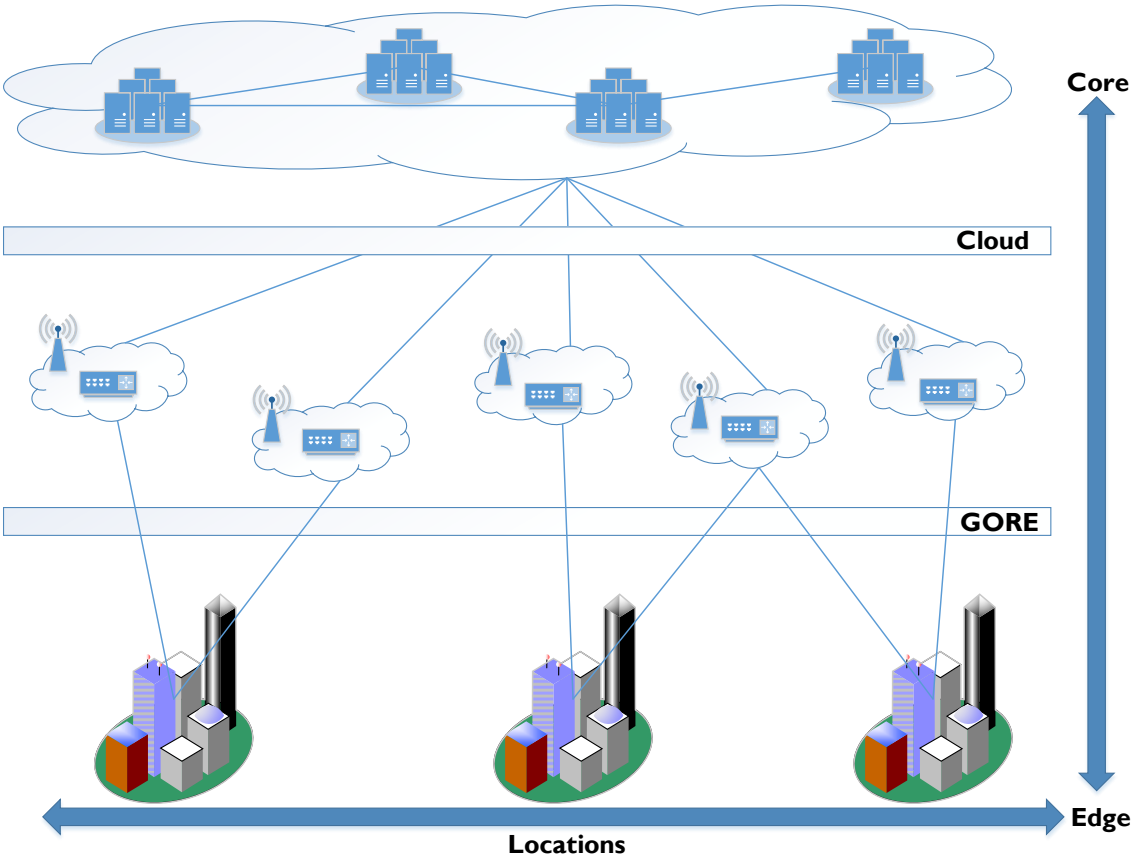

Figure 7.1: GORE at the Edge Below the Cloud.

sistance of the policy management module which is our second contribution in this work. Once a request is evaluated by the policy management module, and a decision made, services are either provisioned to a user or denied. The provisioning of services indicates that the Data Aggregation module will collect relevant data from the IoT device, the centralized cloud data centers, and adjacent localized GNs, to provide the end-user with accurate and updated information.

While the orchestration layer handles request, it is also designed to parse through the communication protocols that IoTs are and will be designed to handle. This provisioning takes place in the Data API and Distributed Messaging Service module.

Currently, three primary IoT protocols are know, primarily the RPL, CoAP, and the 6LoWPAN protocols. However, the technology market has shown growing trends towards IoT. Furthermore, the vast support toward the development of a uniform IoT communication infrastructure by major companies such as Nvidia, Intel, and Mercedes leads us to believe that newer, more robust, communication protocols are on the horizon. This places the GORE infrastructure at a prime location for service provisioning. As, not only does the proposed infrastructure have a uniform communication capability, but it also accommodates for provisioning of resources such as networking, computation, and short-term storage for geographically located IoT devices that require real-time alerts about events in their surrounding. Fig 7.1 illustrates the geographically distributed nature envisioned for the GORE infrastructure. Although this work related the GORE architecture to supporting Smart Transportation Systems, the flexibility of the architecture, allows a GORE infrastructure to function as decentralized nodes intelligently communicating with centralized data centers only utilizing resources as required. By placing GORE environments at the edge of the network, development of Smart Cities, Grids, and Buildings, can be made a reality.

Finally, a prominent and unique contribution of the GORE infrastructure is independence. While the different layers of the infrastructure have multiple interconnected components, each component has been specifically designed to function independently as a module. For example, if an owner of a GN would like to utilize his own customized Data API, the company would utilize the Gateway-based APIs in the Resource Interface Layer and create a custom API for data processing and recognition. Another example could be a tenant who would like to utilize another company's Policy Management module. They would then replace the current policy management framework module with their customized module. This flexibility allows for other computing models such as Fog Computing and Edge Computing to be easily

integrated into a large decentralized environment, thus making GORE a very elastic infrastructure.

7.1.2 Policy Management Framework Contributions

In addition to the core functionalities of the orchestration layer in a GORE architecture, the integration of the policy management module is the second vital contribution to this work. The policy management module is an independent component designed to reside within the orchestration layer of the GORE architecture. This module evaluates user requests against specified policies for an application and the node hosting the application. This research designed a well-defined policy specification with a definite rule schema that will allow for efficient rule and policy evaluation by the policy decision engine. The goal of the policy management module is to deliver services to users in real-time.

In addition to the rule and policy specifications, the policy management module also consists of independent sub-modules for policy and rule analysis to determine conflicts. These conflicts arise when multiple administrators for an application create rules similar or redundant to each other for a single application. Therefore, this research proposed two unique algorithms. The first algorithm utilized a segmentation-based approach and set operations to analyze rules in a policy and segregate the conflicting rules into segments for resolution. The second algorithm took the identified segments and extracted the unique attributes to determine the underlying conflicting conditions and then, based on the administrators resolution preference, and once again utilizing set-operations, the conflicting rules were resolved to produce a conflict-free policy. These policies could then be stored in the policy repository of the policy management module for usage by the decision engine.

The addition of the policy management framework to the GORE infrastructure

reduces the risk of conflicts occurring when a GN or GI is servicing a client's request. Additionally, the framework mitigates the risk of data integrity and accessibility being violated by an unauthorized device. Finally, the framework prevents the occurrence of redundant policies from being uploaded to the policy decision engine, thus ensuring that IoT devices are met with minimal latency when requesting services.

REFERENCES

Andersen, H. R., "An introduction to binary decision diagrams", (1997).

Andy Davis, W. E. W., Jay Parikh, "Edgecomputing: Extending enterprise applications to the edge of the internet", ACM conference on World Wide Web (2004).

Bell, D., *The Coming Of Post-industrial Society* (Basic Books, 2008), URL `http://books.google.com/books?id=q6_56x5tB7gC\&printsec=frontcover\&source=gbs_ge_summary_r\&cad=0#v=onepage\&q\&f=false`.

Belshe, M. and R. Peon, "Spdy protocol", (2012).

Bertino, E., C. Brodie, S. Calo, L. Cranor, C. Karat, J. Karat, N. Li, D. Lin, J. Lobo, Q. Ni, P. Rao and X. Wang, "Analysis of privacy and security policies", IBM Journal of Research and Development **53**, 2, 3:1–3:18 (2009).

Bonomi, F., R. Milito, P. Natarajan and J. Zhu, "Fog computing: A platform for internet of things and analytics", in "Big Data and Internet of Things: A Roadmap for Smart Environments", edited by N. Bessis and C. Dobre, vol. 546 of *Studies in Computational Intelligence*, pp. 169–186 (Springer International Publishing, 2014).

Bonomi, F., R. Milito, J. Zhu and S. Addepalli, "Fog computing and its role in the internet of things", in "Proceedings of the first edition of the MCC workshop on Mobile cloud computing", pp. 13–16 (ACM, 2012).

Bradley, J. H. J., Joseph; Barbier, "Embracing the internet of everything to capture your share of $14.4 trillion", White paper, Cisco Inc., URL `http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf` (2013).

Bureau, U. C., URL `http://www.census.gov/population/www/censusdata/hiscendata.html` (1990).

Cisco, "Big data and analytics: The fuel of the internet of everything economy", Cisco Consulting Services URL `http://tinyurl.com/ngl9rzk` (2013).

Cisco, "Cisco fog computing with iox", Tech. rep., CISCO, URL `http://www.cisco.com/web/solutions/trends/iot/cisco-fog-computing-with-iox.pdf` (2014).

Daly, J., URL `http://www.statetechmagazine.com/article/2013/09/13-cloud-computing-stats-cios` (2013).

DOT, U., URL `http://www.its.dot.gov/research.htm` (2014).

Evans, D., "The internet of everything: How more relevant and valuable connections will change the world", Cisco IBSG, URL `http://www.cisco.com/web/about/ac79/docs/innov/IoE.pdf` (2012).

Hu, H., G.-J. Ahn and K. Kulkarni, "Detecting and resolving firewall policy anomalies", Dependable and Secure Computing, IEEE Transactions on **9**, 3, 318–331 (2012).

Hu, H., G.-J. Ahn and K. Kulkarni, "Discovery and resolution of anomalies in web access control policies", Dependable and Secure Computing, IEEE Transactions on **10**, 6, 341–354 (2013).

International, K., "Accelerating innovation: the power of the crowd", KPMG Research URL `http://www.kpmg.com/Global/en/IssuesAndInsights/ArticlesPublications/accelerating-innovation/Documents/ehealth-implementation.pdf` (2012).

Lewis, G., S. Echeverria, S. Simanta, B. Bradshaw and J. Root, "Tactical cloudlets: Moving cloud computing to the edge", in "Military Communications Conference (MILCOM), 2014 IEEE", pp. 1440–1446 (2014).

Madsen, H., B. Burtschy, G. Albeanu and F. Popentiu-Vladicescu, *Reliability in the utility computing era: Towards reliable Fog computing*, pp. 43–46 (IEEE, 2013).

Mansor, A., W. Kadir, T. Anwar and S. Sahibuddin, "Analysis of adaptive policy-based approach to avoid policy conflicts", in "Software Engineering Conference (APSEC), 2012 19th Asia-Pacific", vol. 1, pp. 754–759 (2012).

McHugh, J., "Radio-frequency chips are retail nirvana. they're the end of privacy. they're the mark of the beast. inside the tag-and-track supermarket of the future.", URL `http://archive.wired.com/wired/archive/12.07/shoppers.html` (2004).

Mora, R. D. L., "Cisco iox: An application enablement framework for the internet of things", URL `http://tinyurl.com/cisco-iox-iot` (2014).

Press, G., "A very short history of big data", Forbes (2013).

Putzolu, D., "Policy-based network management system using dynamic policy generation", URL `http://www.google.com/patents/US6578076`, uS Patent 6,578,076 (2003).

Qlik.com, "Qlikview and big data: have it your way", Tech. rep., Qlik (2014).

RainStor, URL `http://rainstor.com/2013_new/wp-content/uploads/2013/04/RainStor-For-Hadoop-Solution-Brief.pdf` (2013).

Rissanen, E., "extensible access control markup language (xacml) version 3.0", Oasis Standard (2013).

Rubio-Medrano, C., C. D'Souza and G.-J. Ahn, "Supporting secure collaborations with attribute-based access control", in "Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on", pp. 525–530 (2013).

Stafford-Fraser, Q., "The trojan room coffee pot", non-technical biography URL `http://www.cl.cam.ac.uk/coffee/qsf/coffee.html` (1993).

Teo, L. and G.-J. Ahn, "Towards effective security policy management for heterogeneous network environments", in "Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on", pp. 241–245 (2007).

UNECE, URL http://www1.unece.org/stat/platform/display/msis/Big+Data (2013).

Winshuttle, URL http://www.winshuttle.com/big-data-timeline/ (2014).

Wu, Z. and Y. Liu, "Dynamic policy conflict analysis for collaborative web services", in "Network and Service Management (CNSM), 2010 International Conference on", pp. 338–341 (2010).