2012

Applications of Ontology in Heterogeneous Multi-tier Networks for Network Management

Lisa M. Frye
Lehigh University

APPLICATIONS OF ONTOLOGY IN HETEROGENEOUS MULTI-TIER NETWORKS FOR NETWORK MANAGEMENT

by

Lisa M. Frye

A Dissertation

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Computer Science

Lehigh University

January 2012

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Lisa M. Frye
Applications of Ontology in Heterogeneous Multi-tier Networks for Network Management

Defense Date

Accepted Date

Dissertation Director
Liang Cheng
Associate Professor
Lehigh University

Committee Members:

Jeff Heflin
Associate Professor
Lehigh University

Roger Nagel
H. Wagner Professor
Lehigh University

Kent Wires
Principal Software Architect
LSI Corporation

Acknowledgements

I would like to thank my advisor, Dr. Liang Cheng, for his support and guidance throughout my dissertation work. He provided essential input and awareness that facilitated higher quality research and publications.

I am appreciative of the constructive feedback and valuable insights provided by the other committee members: Dr. Jeff Heflin, Dr. Roger Nagel, and Dr. Kent Wires. I am thankful to them for their time, contributions and encouragement during my dissertation process.

The input of my fellow members of the Laboratory Of Networking Group (LONGLAB) at Lehigh University throughout this process was beneficial and I am appreciative of their contributions. Specifically, I would like to thank Zhongliang Liang for his collaboration on the development of the Analytical Model.

I am grateful to my colleagues at Kutztown University that sacrificed their time to provide assistance. I would especially like to thank Dr. Randy Kaplan for his guidance and motivation. I would also like to thank Dr. Dale Parson and Dr. Daniel Spiegel for being readers.

Last, but certainly not least, I would like to thank all my family and friends that provided support and love through this arduous process. I would especially like to thank my mother and grandmother for their love and support and for teaching me when I was young that I could achieve great things. I would also like to thank HL for the love and understanding provided during this evolution and for reminding me that I am strong.

Table of Contents

List of Tables

List of Figures

Abstract

Networks are used by millions of people and have become an integral part of daily life. Network managers strive to achieve 99.9% uptime for their networks. A suite of monitoring and maintenance tools that are used by network managers make up the primary method for managing the network. Networks have been constantly evolving over the past decades. Recent trends demand heterogeneous networks consisting of a variety of devices from various manufacturers. The devices in these heterogeneous networks may consist of traditional wired devices, ad hoc devices or Wireless Sensor Network (WSN) devices. Combined into a single network infrastructure, each of these device types forms a tier within the network resulting in a multi-tier network. If all device types are present, the network will consist of three-tiers, one each for wired devices, ad hoc devices, and WSN devices. Network management of Heterogeneous Multi-tier Networks (HMNs) is both a necessary and complex task for the seamless interoperability of managing the diversities of device types.

One aspect of network management that is of particular interest in today’s climate of increasing attacks and security threats is security management. There are many components to security management, including virus protection, firewalls, and intrusion detection. Attacks are constantly evolving as they adapt to existing security measures making intrusion detection more difficult. Adding to this complexity is the volume of data on networks making any non-automated data analysis task to identify intrusions nearly impossible.

The primary goal of this research is two-fold. The first goal is to provide a Network Management System (NMS) for HMNs. One contribution toward this goal is to address this arduous task by providing descriptions of the devices in the form of a knowledge-based ontology. This integrated Heterogeneous Multi-tier Network Management System (HMNMS) allows a network manager to manage devices of all tiers in a HMN seamlessly and enables automating the data analysis process. A framework was designed and developed for managing HMNs based on ontological descriptions and related algorithms and a prototype HMNMS was built to prove the feasibility of this goal. Another contribution of this goal is the development and verification of an analytical model based on queuing theory that is used to conduct a performance analysis of a HMN. The performance analysis using the analytical model showed the bottleneck to be a gateway node and not the HMNMS in a representative HMN.

The second goal is to develop a formal representation of complex attacks using ontology. This will automate some of the data analysis allowing for the detection of more complex attacks as well as attack attempts. The development of a formal representation using ontology based on generalized attack trees for complex attacks, which provided flexibility and extendibility, is one contribution toward this goal. Furthermore, utilizing network traffic data in the formal representation and detection process provided a way to analyze all traffic data and not just data exploiting existing vulnerabilities. A result of this process led to the detection of additional complex attacks and attack attempts. A set of heuristics was designed and developed based on the formal ontological representation as a second contribution of this goal. A prototype system was constructed to validate the feasibility of using the formal representation and heuristics to detect complex attacks and attack attempts. The new system detected more complex attacks as well as attack attempts than a current state-of-the-art system.

In summary, as networks evolve in complexity and sophistication, a greater need emerges to develop new protocols and mechanisms to manage and protect them. Ontology is utilized in a NMS to manage HMNs, particularly in configuration and security management. Through the use of ontology, interoperability and inference can be leveraged to provide a common management system for a network consisting of heterogeneous nodes and multiple node types, such as wired networks, ad hoc networks, and Wireless Sensor Networks. The main contribution of this work is taking advantage of ontology in the network management domain to add reasoning to management tasks, specifically configuration and security management, consequently reducing the amount of manual analysis required to complete these tasks. The use of this technology will provide additional data analysis to network managers in simplifying management tasks in order to achieve the goal of 99.9% uptime.

Chapter 1 **Introduction**

There are many types of networks, from wired to wireless, wide-area to local, unsecured to strictly secured, static to dynamic, and others. The most common network type is the wired network, consisting mostly of static nodes, such as desktops, servers and printers. These static nodes are interconnected using network devices, such as switches and routers. There are many manufacturers of network devices, including Cisco and Nortel. The different types and manufacturers of nodes in a wired network, together, create a heterogeneous network.

The wired network and the Internet have evolved with the advancement of technology, such as laptops, PDAs, smart phones, e.g. the iPhone, Android and Blackberry and computer surfaces, e.g. the iPad and tablet. These devices have increased the popularity of mobility with the ability to connect to the Internet. In order for these devices to communicate, they must be connected to a network. Often the devices will connect to each other forming their own network. This type of a network is an ad hoc network (AHN). In order for devices on an AHN to access the vast amounts of information available on the Internet, the AHN must be able to connect to the Internet. When this type of connection occurs, a heterogeneous two-tier network is created, with one tier being a wired network, possibly the Internet, and the other tier being an AHN. The network will often be heterogeneous, which, by definition, consists of many different types of devices.

Wireless sensor networks (WSNs) are a type of AHN but require different protocols and applications than traditional AHNs due to the disparate characteristics and constraints of WSNs. The main characteristics of WSNs that make them unique and require the use of new protocols and applications are that they consist of many small sensors that are densely deployed, have limited resources and often have little human interaction post-deployment. There are innovative uses of WSNs, including environment sensing, military scenarios, habitat monitoring, structure monitoring, and first responder situations. Likewise, there are also challenges associated with WSNs. The primary challenge is energy consumption. Connecting a WSN to an AHN and/or a wired network creates an additional tier within the network, resulting in a contemporary type of two-tier network, or possibly a three-tier network. Multi-tier networks, with heterogeneous devices, are known as Heterogeneous Multi-tier Networks (HMNs). An example of a HMN is depicted in Fig. 1.1. In this figure, a Nortel and Cisco device is either a switch or a router. An ad hoc device is any device that is part of an AHN, typically a laptop, tablet device, smart phone, or any other ad hoc device.

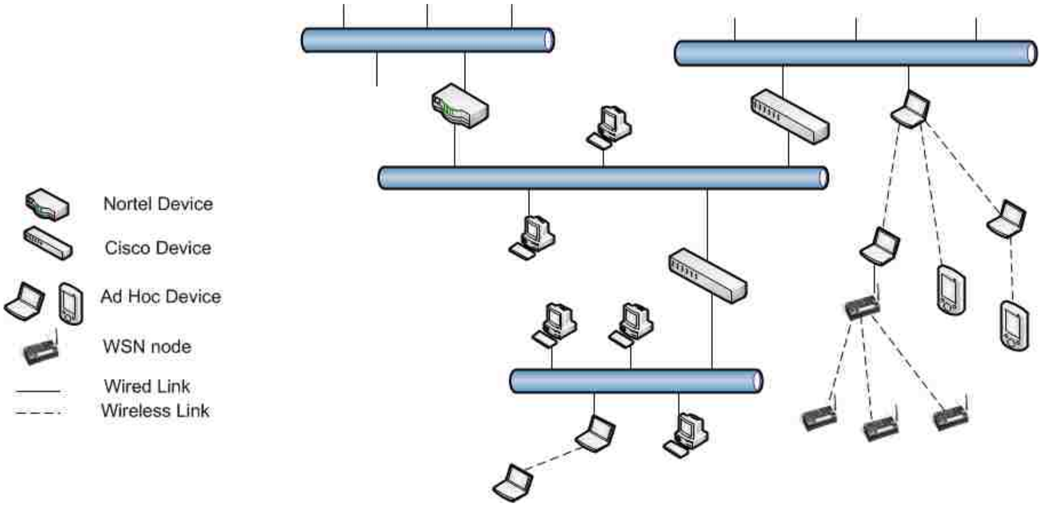


Figure 1.1: An example of a Heterogeneous Multi-tier Network.

Network management is required for all types of networks, including wired, AHN and WSN. Network management is an essential aspect of all networks that includes a suite of tools, protocols, and frameworks used to assist a network manager with monitoring and maintaining networks and all network components. One of the most vital aspects of network management is availability; users expect the network to be available seven days a week, twenty-four hours a day. This demand by users makes network management imperative.

HMNs, with different management systems, make network management a difficult task. Each type of network will typically require its own network management system. Even networks of the same type may require multiple network management systems due to the heterogeneous devices deployed and their proprietary nature. A network management system that can manage devices from various manufacturers and different network types would simplify this task.

To develop one network management system, the existing systems, or models, must be merged or mapped into a single model. A single model will ensure only one language interpreter with an integrated definition of all network elements and their associated behavior. In order to develop one network management system, with one language interpreter, it is necessary to understand the syntax and the semantics within each management system.

Ontology [1, 2] is an area of research that can assist in the mapping of all network management systems into a single system for easier management of a HMN. This is accomplished by using ontology constructs to indicate which elements in one management system are equivalent to elements in another management system. Providing descriptions of the subcomponents in the form of a knowledge-based ontology is one way to address the arduous task of managing these networks. Our primary goal is to create a framework for managing these networks based on ontological descriptions and related algorithms.

To map multiple ontologies into a single domain, mapping rules are developed that will translate data from each management system. To develop the mapping rules, it is necessary to understand the semantics of data within each domain or model. The ontology based management system developed in this research can be used to manage a one-tier network, which can be a wired network, AHN, or WSN, or a multi-tier network consisting of a combination of these.

A secondary goal in the management of heterogeneous networks is to recognize and act on complex attacks as they may occur. Attacks can take the form of sequences of events that result in a complex attack. To date, this problem has only been addressed on a limited basis due to the heterogeneous nature of networks and the infinite possibilities of sequences that may result in a complex attack. Our ontological representation will use collected knowledge about attacks so that a management system can proactively detect and act upon complex attacks. We will demonstrate this enhanced complex capability in our management system on our data to detect a greater volume of complex attacks as well as attempted attacks. Attack detection in this research is based on all network traffic rather than just on vulnerability data allowing for the detection of a wider variety of complex attacks and attack attempts.

A security attack against a network device may cause it to work incorrectly or not at all. Depending on which device in the network failed, the attack would cause at least a portion of the network to fail. If a network device fails and that device is connected a subnet to the network, then the entire subnet would become disconnected from the network. If the device attacked was a core network device, it may affect the entire network, making the network unusable.

It is imperative to prevent such failures from occurring so that 100% uptime of all network nodes may be maintained. To prevent device failure due to a security attack, a system must be deployed that will detect network attacks or intrusions. Such a system is an Intrusion Detection System (IDS), which will create an alarm when an intrusion is detected. This is known as a true positive in an IDS, meaning that an alarm is generated when an attack is detected, and there is indeed an attack. Many IDSs will detect an intrusion in the network by analyzing existing vulnerabilities of the deployed devices, primarily deployed hosts (not network devices). Others will scan network traffic and identify possible attacks to the network. The attacks identified by these IDSs are simple attacks; the IDS will raise an alarm when a single attack type is identified in the network traffic. A single attack may in itself be meaningful but this does not necessarily preclude the possibility of a larger context for the single attack.

To be able to provide the best IDS solution, existing vulnerabilities in all deployed devices are analyzed, including network devices, and network traffic is scanned to identify all types of attacks, including multi-phase attacks. This process is accomplished by including all deployed nodes, hosts and network devices, in the vulnerability analysis phase, and analyzing simple attack alarms to identify those that are a step in a multi-phase or complex attack.

Another step in the IDS is to identify potential attacks. The network manager should not only be concerned with attacks against nodes that are vulnerable, but should also monitor for any attack attempt by an intruder. This is vital because a network manager cannot predict precisely which applications users on the network may install or deploy. For example, if an intruder is attempting to circumvent a web services vulnerability but there are no vulnerable systems on the network, then the attack attempt is unsuccessful. The same attack may be successful in the future due to a user installing a new web server that is vulnerable to that particular attack. To make the network more resistant to successful attacks, the network manager should analyze all attack attempts against it.

The final phase of the IDS is to identify a remedy for the attack. If there was an unsuccessful attack, the remedy would simply be the identification of the reason it was not successful so this attack type will continue to be unsuccessful, even with the deployment of new devices or additional services on the network. For a successful attack, the remedy will ensure failure in the future. This may require a patch to a host or a reconfiguration of a service or node. The reconfiguration may be dynamic, performed by the IDS, or it may require manual intervention based on remedy information included in the alarm to the network manager. In both cases, the application of the remedy will make the network less susceptible to future attacks. Along with the remedy, it may be desirable to place the source of the attack on a blacklist, which is a list containing addresses that are forbidden from sending data to the network.

By ensuring that network devices are secure and resilient, the network will remain operational for a longer period of time, helping to meet the goal of maximizing network availability. This requires the ability to detect when a network node fails and possibly perform a reconfiguration to minimize the failure. This may be a reconfiguration of the failed node to correct the problem and restore its functionality in the network, or the reconfiguration of the network or subnet to bypass the failed node. The ability to identify a possible failure or a cause of a failure, and then a possible remedy, will ensure continued operation of the network, regardless of the network type.

In the event that the attack cannot be detected in real time, an IDS can still provide valuable information. As demonstrated by the information security life cycle illustrated in Fig. 1.2, security is an on-going process. The steps in the information security life cycle are risk analysis, risk assessment, cost/benefit analysis, implementation, and vulnerability assessment. The cycle begins with risk analysis, which involves identification of the organization’s assets and the vulnerabilities present in each asset. The next step, risk assessment, determines the threats against the identified assets, the probability that those threats will occur, and the consequences of each threat occurring. Cost/benefit analysis is the third step. It is used to determine the best controls to implement based on the ones that address the identified threats at an appropriate cost. The appropriate cost depends on the organization and its goals. The implementation step, which is the fourth step, is the deployment of the identified controls during the cost/benefit analysis. The vulnerability assessment is the final step. It is used to determine if the implemented controls are working appropriately. At this point, the risk assessment is completed again and the cycle begins once again. Due to the fact that the assets, threats and controls are constantly being evaluated, even if an attack is detected post-success, the information about the attack can still be very useful in future security detection and intervention.

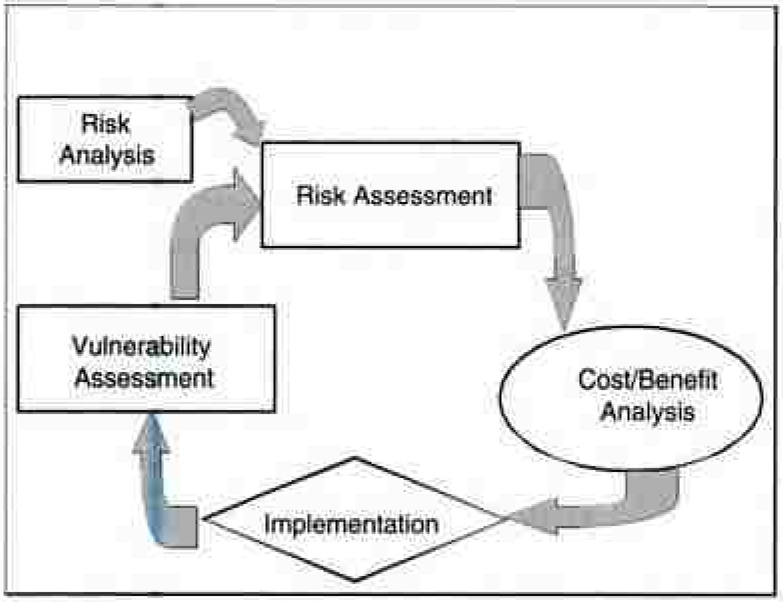


Figure 1.2: The information security life cycle [3].

1.1 Contributions

The goal of this research is to provide a method to manage Heterogeneous Multi-tier Networks, which currently does not exist, by designing and developing a framework based on ontological representations. The contributions are:

1. The design and development of an ontology based Network Management System (NMS) consisting of an adaptable knowledge base structure that significantly enhances the ability to manage HMNs as they evolve in number and complexity.
2. The development and verification of the first analytical model for conducting performance analysis of a HMN.
3. The design and development of an ontological representation for simple and complex attack types based on generalized attack trees facilitating

improvements in attack detection and allowing for augmentation of basic attack knowledge.

4. Improve on the expressivity of complex attack reasoning and recognition by augmenting the ontological representation with a set of extensible heuristics designed for this purpose.

The first contribution of this research is the design and development of an **ontology based Network Management System (NMS) consisting of an adaptable knowledge base structure** that significantly enhances the ability to manage HMNs as they evolve in number and complexity. This NMS addresses the challenges new technologies and dynamic components present to heterogeneous network managers. It provides seamless integration of support to manage Heterogeneous Multi-tier Networks, even as they evolve. An ontology based approach to network management is designed and developed so it can be implemented by others and demonstrated in our prototype system. The rational and advantages of the ontology based approach are outlined. A prototype ontology based NMS is built and an existence proof is provided that shows the feasibility and performance goals are achievable. An example is provided that shows this approach to network management is an n:1 improvement in the toolset required for management of a HMN, where n is the number of different device types in the network.

The second contribution is the **development and verification of the first analytical model for conducting performance analysis of a HMN**. The analytical model for a HMN is developed based on queuing theory. A performance analysis of a HMN is conducted to verify the model and identify bottlenecks. The analytical model is then utilized to prove that the bottleneck in a Heterogeneous Two-tier Network is the ad hoc gateway and not the Heterogeneous Multi-tier Network Management System (HMNMS).

The third contribution is the **design and development of an ontological representation for simple and complex attack types** based on generalized attack trees facilitating improvements in attack detection and allowing for augmentation of basic attack knowledge. The ontological representation provides more flexibility because its declarative representation allows for augmentation without impacting other aspects of the system. This allows it to be extended by others doing related research therefore extending the knowledge and enabling the detection of evolving attack strategies. Generalized attack trees are defined for complex attacks based on the analysis of attack patterns. The utilization of traffic data in developing the formal representation and its advantages are described. The formal representation of the complex attacks based on traffic data is developed using ontology, which provides flexibility over a programmatic approach. This representation enables the knowledge to be extended by others doing related research therefore extending the knowledge and surviving the evolution of complex attacks.

The fourth and final contribution of this research is to **improve on the expressivity of complex attack reasoning and recognition by augmenting the ontological representation with a set of extensible heuristics** designed for this purpose. There is a trade-off between the expressivity in knowledge representation languages and the computational complexity. A highly expressive language is used for the ontological representation but some expressive limitations exist that prevent the representation of all complex attacks. The heuristics are developed to add the necessary expressivity using the ontological representation. These heuristics are expressed as queries in SPARQL, a standard query language, enabling easy modification and addition of rules to detect additional complex attacks. The ontology based set of heuristics is developed to allow for implementation. A flexible prototype system is developed to show the viability of using the heuristics to detect complex attacks and attack attempts. In the analysis of data, results showed the prototype system detected more complex attacks and attack attempts than a current state-of-the-art system used for comparison.

1.2 Dissertation Roadmap

The dissertation is written to provide the reader with a progressive flow of this research. Chapter 2 provides background information on the technology used in the research. Related works for the various aspects of the research is provided in Chapter 3. The next four chapters provide details for each of the four contributions. Chapter 4 describes the developed HMNMS, including the theoretical basis and ontology based approach. This chapter includes preliminary results for the prototype HMNMS deployed in experimental and live networks. The analytical model for performance analysis of a HMN and results of an analysis of an experimental network are explained in Chapter 5. Chapter 6 explains the design and development of a formal representation for complex attacks and attack attempts. The approach and development of the formal representation are also described in this chapter. Chapter 7 describes the design and development of a set of heuristics based on the formal ontological representation. As such, the definition of a set of heuristics and the development of a prototype system using the set of heuristics are explained. Chapter 8 includes conclusions and future work for this research.

Chapter 2 **Background Information**

2.1 Network Management

The key elements of network management are a management station, management agent, an information base, and a protocol. The management station is typically a desktop or laptop that collects the data from the managed devices. In order for devices to be managed, there must be software installed on each device to communicate with the management station. This software is the management agent. The information base is the data that is to be collected by the various types of managed devices. The communication between the management station and the management agents is through a management protocol. The management protocol will ensure that the management station and agents are using the same syntax and semantics for exchanging messages.

The International Organization for Standardization (ISO) created a network management model to aid in understanding the functionality of network management. This network management model consists of five functional areas: 1) fault management, 2) configuration management, 3) performance management, 4) security management and 5) accounting management.

The ability to identify problems in the network is the primary goal of fault management. The steps in fault management include: a) determining a problem exists, 2) isolating the problem, and 3) fixing the problem, if possible. When a fault occurs, the network manager receives an alarm, which may be in the form of a log file entry, an E-mail message, an SMS message, a page, or an entry in the network management system.

The number of faults occurring in a network are usually too numerous for the network manager to address individually. In order for the network manager to successfully address the faults in a systematic manner, a prioritization of the faults is crucial.

Configuration management includes setting up, monitoring and controlling network devices. To assist with many of the other network management tasks, an inventory of all network devices must be maintained. This inventory should include the devices deployed and their characteristics, such as the name, network address, location, both physical and logical, and the current configuration. The inventory and collection tasks are often referred to as topology management. It may also include a physical or logical map of the network. The information for the inventory should be collected on a periodic basis, either manually or automatically. A common collection method is called autodiscovery. Autodiscovery is a process that runs on a network management system and periodically detects all installed network devices. It reports back to the management station each device found and some of the device’s characteristics. While this process is an effective automated tool for collecting network inventory information, it is bandwidth-intensive and is not recommended for bandwidth-constrained networks, such as WSNs.

A critical aspect of network management is procuring the utilization of the network devices and links. This is the job of performance management. Having this information about the network components will assist the network manager in troubleshooting, identifying bottlenecks and capacity planning. The type of the component will indicate the utilization information that is important and may include utilization of the CPU or network card. Some of the specific items of interest in performance management are packet forwarding rate, error rate, and packets queued.

Security management is often a challenging task to complete and is distinct from operating system, physical, and application security. Security management requires restricting access to information on the network and the network components to those entitled to it. The primary function of security management is controlling access points to data that is stored on devices attached to the network.

While performance management tracks the utilization of network components, accounting management tracks the utilization for each user. This includes the utilization by each user for the various network resources, including network devices, links, servers and storage devices. The original reason for the inclusion of this functional area by ISO was to allow organizations to bill users for their usage of the network and its resources. While this is no longer a common practice within organizations, the information gathered about users is still useful to a network manager, particularly to aid in establishing metrics and quotas. It is also helpful to allow proper allocation of network resources. User utilization may also overlap with security management. This process allows the network manager to understand typical user behavior; if atypical behavior is detected, then it may indicate a security breach or intrusion.

Network management of AHNs and WSNs is more difficult in general due to their dynamic nature and the limited resources of the devices. The five functional areas identified by ISO are a part of the network management of AHNs and WSNs. These areas may be modified or augmented for proper management of these network types. For instance, network coverage and connectivity are a part of performance management. The nature of ad hoc networks makes security management in AHNs and WSNs more difficult. This is a result of the use of wireless communication, which is more difficult to secure, and the resource limitations of the devices.

An important aspect of AHNs and WSNs, which is not part of wired network management, is energy management. Energy management may be regarded as a separate functional area or encompassed in several of the functional areas. Including energy management as a part of some of the other functional areas is often done because of the overlap with the different areas and energy management. Energy management is often included within the areas of configuration management, fault management and performance management. It is considered part of these areas, instead of its own area, because of its close connection to these tasks. When nodes run low on or out of energy, it impacts these other areas. For example, in topology management, which is a part of configuration management, if a node runs out of energy, it is no longer a part of the network. If this node was a part of the routing protocol or a gateway node, then the network topology will change. This may also generate a fault or impact the performance of the network, thus demonstrating the reason to include energy management with fault and performance management.

2.1.1 SNMP

The standard network management protocol for wired networks is Simple Network Management Protocol (SNMP) [4]. SNMP is considered simple because it is based on two commands, fetch and store. All operations are implemented using these two commands. The basic operation of SNMP is the management station requesting data from managed devices via the fetch command. The devices will return stored data to the management station in response to these requests. The store command is used by the management station to set values by saving a specified value to an attribute in the device. The device attributes are called objects.

All the objects SNMP can access require a definition, including a unique name. The management station and management agent must agree on the object names so there is a common vocabulary for communication. The set of all objects that SNMP can access is defined by the Management Information Base (MIB). By separating the object definitions from the management software, new items can be added to the MIB while maintaining the same software.

Along with the object specification, the MIB also defines any object groupings and relationships between managed objects. The object definitions are specified using the Structure of Management Information (SMI). SMI is a subset of the Abstract Syntax Notation One (ASN.1), which is a standard for describing data structures.

The names specified for all managed objects are taken from the Object Identifier Namespace [5], which is administered by ISO (International Organization of Standards) and ITU. The Object Identifier Namespace describes a namespace for arbitrary objects and is not dedicated to network management. Examples of objects that can be referenced using the Object Identifier Namespace are a company, a project, an encryption algorithm, a file format, and a SNMP MIB.

The Object Identifier Namespace is a hierarchical structure with each node specified with a unique name and number. The Object Identifier (OID) is the sequence of the numeric labels of the nodes in the path from the root to the object. A part of the namespace is provided in Fig. 2.1, which is the subtree for internet management. Following the nodes from the root to the internet management node produces an OID of

1.3.6.1.2. As illustrated in Fig. 2.1, *mib* is the node below the *internet management* node. Below the *mib* node is a node for each MIB category. These categories are *system*, *interfaces*, *at* (this one is deprecated and only remains for compatibility), *ip*, *icmp*, *tcp*, *udp*, *egp*, *transmission*, and *snmp*.

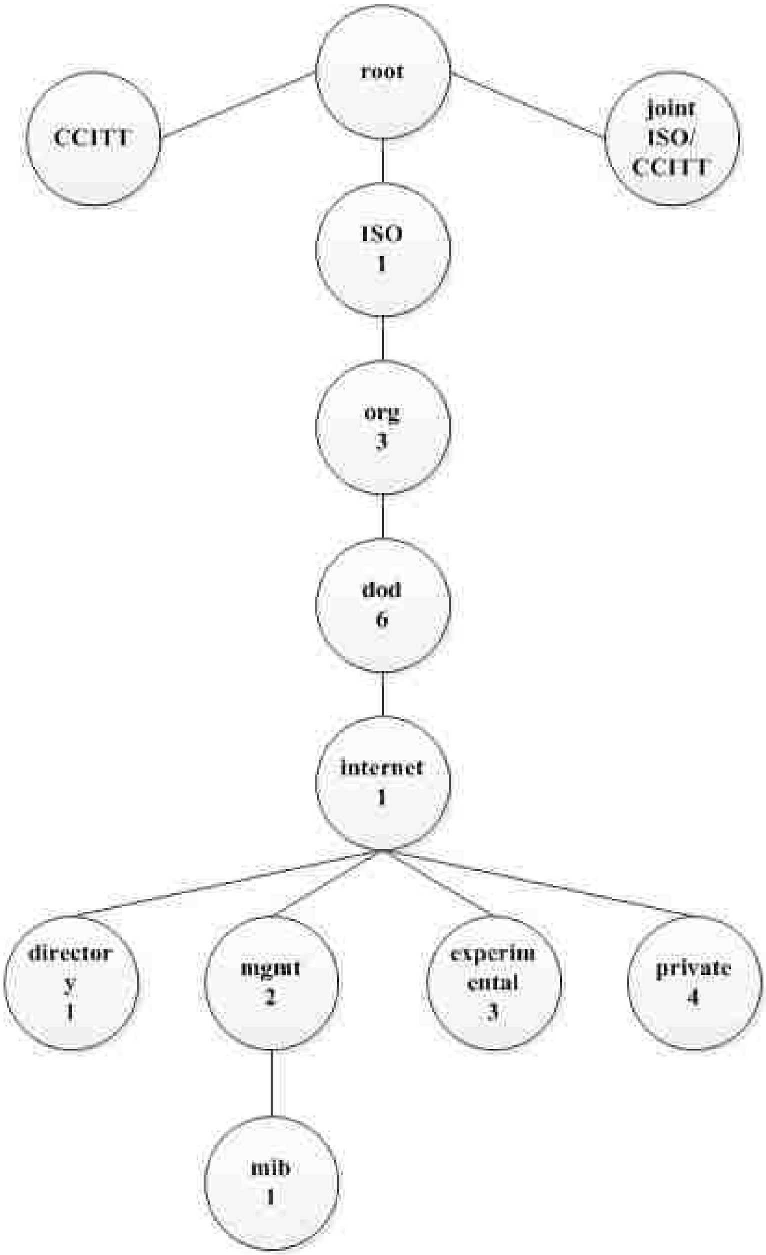


Figure 2.1: The Internet management subtree of the Object Identifier Namespace (OID).

The MIB specifies the data each network device type, such as switches and routers, must maintain. The unique names for each object are defined in the MIB, as well as the meanings of each and the operations allowed on each. An example of an object definition is provided in Fig. 2.2. This example defines an object called *sysName*, which stores a name assigned by the network manager for the managed device. The notation defines the syntax, access permissions, status and a brief description of the object. The {system 5} notation indicates that it is a child node of the system node and the node has a numeric value of 5. This value is used to specify the object's OID. Since it is a child of the *system* node, which has an OID of 1.3.6.1.2.1.1, the OID for *sysName* is 1.3.6.1.2.1.1.5.

```
sysName OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
            "An administratively-assigned name for this managed
            node.  By convention, this is the node's
             fully-qualified domain name.  If the name is unknown,
             the value is the zero-length string."
    ::= { system 5 }
```

Figure 2.2: An example of an object definition in the MIB for SNMP.

2.2 Ontology

Originating in the field of philosophy, ontology is now being used and researched in many other fields, including computer science. In computer science, ontology is a data model representing the knowledge in the specified domain, as well as the relationships between this knowledge. Ontologies define "a set of concepts, its taxonomy, interrelation, and the rules that govern these concepts" [6]. Two fields in computer science that benefit from ontology are Artificial Intelligence and the Semantic Web. There has also been research in using ontology in the network field.

The primary benefits of ontology are interoperability and inference. Interoperability provides a way to share knowledge in a domain, which overcomes differences in terminology for the same concept and meaning for the same term. Interoperability of multiple domains is accomplished with ontology mapping. This will map an item in one domain with an item in another domain. Inference allows new knowledge to be learned from existing knowledge. For instance, if it is known that Jordan is Jack's parent and that Jordan is female, then it can be inferred that Jordan is Jack's mother.

Ontology is a declarative approach, which is typically more flexible than a procedural approach. This makes the system more adaptable. Other benefits from ontology are reusability, reliability, shareability, portability, and interoperability [1]. There are also a large set of tools available for ontology, making it easy to define and use.

Ontology can be classified according to various characteristics. One of the possible classifications of ontology is lightweight or heavyweight. This classification depends on the expressiveness of the language used to describe the ontology. A lightweight ontology is represented with a simple taxonomy or hierarchy of the domain concepts. A lightweight ontology can describe concepts, concept relationships, concept properties, and concept taxonomies. A heavyweight ontology attempts to fully describe the domain concepts by including rules, axioms and constraints.

Another classification of ontology is the generalization of the domain concepts. The different types of ontology in this classification are upper, middle, and lower. An upper or foundational ontology [7] defines general concepts for a domain. It would be used to provide a common foundation to be leveraged by other, more specific domain ontologies.

A middle ontology extends an upper ontology and is more specialized in the domain. An ontology for concepts that are very specific in the domain, often for a specific application in the domain, is a lower or application-specific ontology.

2.2.1 Knowledge Representation Languages

To formally represent domain knowledge, a knowledge representation language may be used. There are various knowledge representation languages, including XML, RDF, RDF Schema and the Web Ontology Language (OWL). The languages differ in syntax and expressiveness. Fig. 2.3 illustrates the knowledge representation languages and how they relate to each other. This image is known as the *Semantic Web Stack* and was created by Tim Berners-Lee. Tim Berners-Lee is the inventor of the World Wide Web, the Director of the World Wide Web Consortium (W3C) [8], and the Director of the World Wide Web Foundation [9].

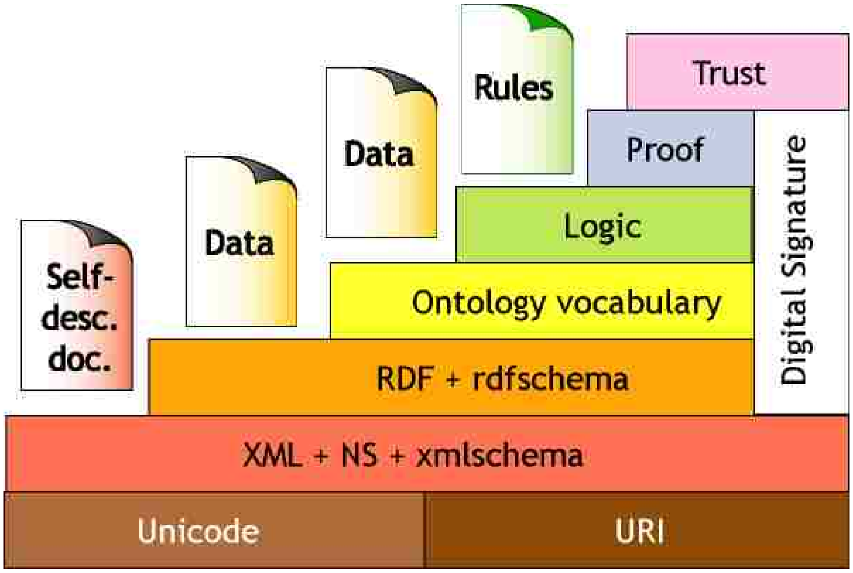


Figure 2.3: Knowledge representation languages [10].

The eXtensible Markup Language (XML) [11] is a language developed to provide a structure of information. This allows information to be represented that is accessible by humans and machines. Relationships among the information can be defined by nesting XML tags. XML provides a structure or syntax to the information but provides no semantics to the information.

A universal language that uses XML-based syntax is the Resource Description Language (RDF) [12]. It allows users to describe resources using their own vocabulary. Resources are described by a set of triples called statements. Each statement consists of a subject, a predicate or property, and an object. The object is the subject's value for the specified property. RDF allows the specification of resources but implies no meaning about them.

One method to make semantic information accessible by a machine is to use RDF Schema (RDFS) [13] by defining the structure of the data. This is also a knowledge representation language that organizes objects into hierarchies. It defines the vocabulary used in RDF data models, specifies the properties that apply to each kind of object, specifies the values for each property, and defines the relationships between objects. The benefit of RDFS is the ability to provide semantic information to machines, but it is limited to the subclass and property hierarchies.

There are several specific limitations of RDFS [14]. First, properties only have local scope so there is no way to specify restrictions that apply to some classes only. Second, it does not provide a way to specify that classes are disjoint. If there are instances that can belong to one class but not another, this is done by saying that classes are disjoint. For instance, mother and father would be disjoint classes because an individual could not be a member of both. Third, new classes cannot be created using Boolean combinations of classes, such as union and intersection. For example, if a class exists for mother and father, then parents would be the intersection of these two classes. Fourth, RDFS does not allow cardinality restrictions. It may be necessary to state that a person can have exactly two parents, which is not possible in RDFS. The last limitation of RDFS is the inability to define special characteristics of properties, such as transitive and inverse. For example, it would not be possible to specify that the “is child of” property is the inverse of the “is parent of” property (if Jack is the child of Jordan, then Jordan is the parent of Jack).

2.2.2 OWL

The most popular ontology language is OWL [15, 16]. OWL is a general purpose ontology language that represents knowledge using RDF triples. It provides a way to express semantic information about resources. OWL allows the user to provide the definition of important domain concepts and the relationships between the concepts through a class hierarchy.

There are three different variants of OWL, OWL Lite, OWL DL, and OWL Full [16]. The differences in these variants are in their expressiveness. OWL Lite is a subset of OWL constructs and also includes restrictions on the use of some of the allowed OWL constructs. For instance, cardinality values can only be 0 or 1 and there is support for intersection only in class definitions. OWL DL is based on description logics and provides computational completeness and decidability. OWL DL supports all OWL constructs but places restrictions on the use of some of the constructs. For example, if a property is declared to be transitive, then it cannot have numeric restrictions placed on it. As another example, classes cannot be individuals of other classes. OWL Full provides the maximum expressiveness and is a superset of RDF. It is the complete OWL language including all OWL constructs with no restrictions on their use. For example, cardinality values can be any value greater than or equal to 0, there is support for intersection, union, complement, and enumeration, and classes can be instances and properties at the same time.

OWL uses a class hierarchy similar to object-oriented programming languages. Members of the class are known as individuals. Individuals may also be referred to as instances. Classes define a way to categorize similar individuals. A *subclassOf* property is used to create the hierarchy. A class that is a sub class of another class will inherit the parent class's properties and will also infer that an individual that is a member of the subclass will also be a member of the parent class. Individuals of a class can be defined using enumeration with all the individuals of that class being defined using *oneOf*. If an individual cannot be a member of two specified classes, these classes are said to be disjoint. Class can be defined to be disjoint using the property *disjointWith*.

Classes can be defined using set operators, including *intersectionOf* and *unionOf*. A class can be the union of two other classes, which results in a class containing individuals that are members of one of the classes. If a class is the intersection of two classes, then it contains all the individuals that are individuals in both classes, but not individuals that are in only one of the two classes or not in any of the two classes. Often *unionOf* and *intersectionOf* are used with property restrictions. For instance, it might be necessary to say that the class daughter is the individuals in the class child that have a value of "female" for the property named gender (indicating male or female).

There are two different types of properties in OWL. A datatype property is an attribute of the individual that will have a value. The value will be a literal of some datatype, such as a string or integer. An object property will have a value that is an individual of another class indicating a relationship between the two individuals. Two property restrictions are *domain* and *range*. Both of these restrictions are specified on object properties and restrict the values of an object property if it relates two individuals. If a class is specified as the *domain* of a property, then the value of individual for the subject of the property must be an individual of the specific class. The same is true for *range* except the range is applied to the value of the property, or the object.

A restriction can be placed on the number of values that can be assigned to a property. This restriction is *cardinality* and it can be a specific cardinality. *MaxCardinality* and *minCardinality* can be used to specify a maximum or minimum cardinality for a property.

A few other property restrictions were used in the research in this dissertation. Several of these restrictions relate one property value to another one. If one property is a *subPropertyOf* another property, then if a subject is related to an object by the specified property, it is also related to the object by the parent property. A property can also be the inverse of another property, using *inverseOf*. For example, *hasChild* could be the *inverseOf hasParent*, since the child is the inverse of parent. Two properties can also be equivalent, using *equivalentProperty*, which creates a synonym property for another property.

Two restrictions were used to specify limitations on the possible value of specified properties. If a property is restricted with *allValuesFrom*, then the value for that property must be an individual from the specified class. This also allows the user to infer information; the object of this property would automatically be an individual of the class specified in the *allValuesFrom* restriction. The *someValuesFrom* simply states that at least one value of that property must be an individual from the specified class.

2.2.3 SPARQL

SPARQL [17] is a query language for RDF and is similar to SQL for databases. It is used to query the knowledge base at the triple level. A SPARQL query can consist of triple patterns, conjunctions, disjunctions, and patterns. There are four forms of a SPARQL query (SELECT, CONSTRUCT, DESCRIBE, ASK). The SELECT, CONSTRUCT and DESCRIBE queries are all used to extract information from the knowledge base with the difference being in how the information is returned. The SELECT query returns the information in table form; CONSTRUCT returns RDF triples and DESCRIBE returns an RDF graph. The ASK query is used to determine if a solution exists and will simply return true or false.

The two primary SPARQL statements are SELECT and INSERT. A SELECT statement will retrieve all information from the knowledge base matching the specific criteria. An INSERT statement will add new statements to the knowledge base. A WHERE clause can be specified in the query to provide criteria to match with the data in the knowledge base. Only data in the knowledge base matching the specified pattern in the WHERE clause will be returned.

To further restrict the solutions returned, the FILTER keyword can be used. This keyword will specify additional criteria to be used to eliminate statements from the solution returned. The filter pattern can be specified using relational and logical operators. Regular expressions can also be specified in the filter pattern using the REGEX keyword.

There are other keywords that can be used to either limit the results returned or specify how the results should be returned. For the purposes of this research, three of these other keywords were employed. The DISTINCT keyword will eliminate any duplicate statements from the solution. If part of the matching pattern is optional, that part of the pattern is restricted with the OPTIONAL keyword. The statements in the solution can be ordered according to specified criteria using ORDER BY.

Aggregates can be used in the solution. Before applying an aggregate the solution set must be grouped. This is accomplished using GROUP BY. If the groups in the solution should be restricted to specific criteria, such as having more than a specified number of statements in each group, the HAVING keyword is used. Some of the aggregates that can be used are count, to return the number of statements in each group, and MIN and MAX, which will return the minimum or maximum value of a specified property in the solution.

ARQ is a SPARQL processor for Jena [18]. ARQ includes a function library consisting of various functions that can be used in SPARQL queries in Jena. A subset of these functions was used in the research in this dissertation. For example, the concat function is used to concatenate several property values that were returned together to form one value.

2.3 Queuing Theory

Queuing theory [19] is used to study the behavior of queues in a system or network [20]. A common queue model used for analysis is the M/M/1 model. The first M represents the type of arrival process to the queue. In the M/M/1 model, the arrival process is a Poisson distribution [21] of arrival requests with a mean rate of λ. A Poisson distribution indicates that the arrival times follow an exponential distribution and the probability that *n* events occur during time *t* is the Poisson distribution. The second M is the service duration of a request, which is exponentially distributed in this model with a mean rate of μ. The 1 indicates there is a single server. The last two values in queuing systems notation are not specified in the M/M/1 model indicating there are an infinite queue length and an infinite number of sources that can produce requests.

Queuing systems can be characterized by several variables, including the mean number of requests, N and the mean wait time or delay, T. The N, T and λ are related by a basic formula known as Little's Theorem [21]. Little's Theorem uses the equation

$$N = \lambda T \qquad (2.1)$$

Little's Theorem demonstrates the obvious conclusion that systems with more requests (large N) will have larger wait times (large T).

There are several equations that can be used to describe a M/M/1 queuing system [22]. The first equation is the traffic intensity, ρ

$$\rho = \frac{\lambda}{\mu} \qquad (2.2)$$

To maintain a stable system and prevent the queues from going to infinity, ρ should be less than one. The mean number of requests in the system, N, can be calculated using the equation

$$N = \frac{\rho}{1-\rho} \qquad (2.3)$$

Substituting the Eq. 2.2 gives the equation

$$N = \frac{\lambda}{\mu-\lambda} \qquad (2.4)$$

The total time spent in the system, T, including the wait and service times, uses the equation

$$T = \frac{N}{\lambda} = \frac{1}{\mu - \lambda} \tag{2.5}$$

This equation is obtained by applying Little's Theorem to Eq. 2.4.

For a network analysis, there is a need to obtain the expected waiting time in the network. This time excludes the transmission time. Little's Theorem provides the equation

$$T = \frac{N}{\lambda} \tag{2.6}$$

The expected waiting time, W, minus the transmission time, results in the equation

$$W = \frac{N}{\lambda} - X = T - X = \frac{1}{\mu} = \frac{\rho}{\mu - \lambda} \tag{2.7}$$

where X is the average transmission time. If N_Q is the average number of packets waiting in the queue, then applying Little's Theorem results in

$$N_Q = \lambda W = \frac{\rho^2}{1 - \rho} \tag{2.8}$$

These equations are all used in queuing theory analysis and were used in the analytical model developed for the network performance analysis conducted in this research.

2.4 Intrusion Detection Systems

One method used to identify attacks is by using an Intrusion Detection System (IDS) [23]. IDSs can be classified using multiple methods. One classification method is based on what the IDS monitors, a host or a network. A host IDS (HIDS) is deployed on a host, or adjacent to a host, to monitor that host for attacks. A Network IDS (NIDS) is deployed on a network and monitors for an attack on the network. It will scan the traffic on the network looking for possible intrusions.

Another classification method for IDSs is how the IDS operates to detect an attack. An IDS can be either signature-based or anomaly-based. In a Signature-based Intrusion Detection System, the system works much like antivirus software and identifies an attack based on whether there is a match against an entry in a signature database. If Signature-based Intrusion Detection System is deployed, it is necessary to maintain a current signature database.

An Anomaly-based Intrusion Detection System is an IDS that looks for behavior that is not considered normal. A baseline must be established and then any behavior that is not within the established parameters may be considered abnormal and a possible attack. An Anomaly-based Intrusion Detection System has the potential to detect a new attack because the longer it runs the more it learns about normal behaviors. However, an Anomaly-based Intrusion Detection System is susceptible to false positives as it is possible to have something look abnormal when it is in fact a normal behavior.

There are advantages and disadvantages to all different types of IDSs. The best solution may be a combination of these different types. This can be done by placing several IDSs throughout the network. These different IDSs may be a combination of HIDS and NIDS, as well as signature- and anomaly-based. This will allow for the best chance of detecting all types of attacks.

Another way to combine the different types of IDSs is to use a new type of IDS, Reasoning-based IDS (RIDS). RIDS utilizes both signatures and anomalies to detect possible intrusions. It integrates both types into one IDS, without the need to manually combine, and then possibly conduct manual analysis on the output of multiple IDSs.

Another advantage of an RIDS is that it may employ advanced reasoning in attack identification. This allows for an efficient and reliable analysis of the data collected from the network to aid in the detection of all types of attacks, including zero-day attacks. A zero-day attack is an attack against vulnerabilities that are unknown. A properly designed RIDS has the ability to detect a multi-phase complex attack such as the one mentioned-above using a combination of port scan or telnet probe and vulnerability exploit.

One way to incorporate advanced reasoning into an RIDS is by using ontology. Ontology allows for the semantics, along with the syntax, of the domain knowledge to be integrated into the system. In the domain of network security, the syntax refers to the signature of an attack, which is the basis of a signature-based IDS. Incorporating semantics allows the RIDS to also make decisions based on the meaning of the data, such as the importance of a ping scan followed by a port scan, within a specified time frame. This may indicate a possible attack, as opposed to seeing a ping scan followed by a network management task. Knowledge of the semantics of the domain, and the domain data, allows the use of inference, which can be used to learn more about the network traffic and possible attacks, both simple and complex.

Chapter 3 **Related Work**

3.1 Network Management

Several management protocols or systems, based on the ISO network management model, have emerged in wired networks, including Internet Engineering Task Force's (IETF) SNMP (Simple Network Management Protocol), ISO's Common Management Information Protocol (CMIP) [24], Distributed Management Task Force's (DMTF) Desktop Management Interface (DMI) [25] and DMTF's Web Based Enterprise Management (WBEM) [26]. Each of these systems is in use today, with SNMP being the most common system used in wired networks. SNMP is a basic request-reply protocol with a smart management station sending requests to a dumb agent on each device to be managed. The agent simply replies to the request with data stored in the device. The only time an agent initiates data transmission is when there is an event that occurs that requires notification to the management station, such as a link down or a power supply failure.

The primary network management protocol in Ad hoc Networks (AHNs) is Ad hoc Network Management Protocol (ANMP) [27]. ANMP is compatible with SNMPv3 and many of its features are based on SNMP. ANMP includes more data items to monitor that are critical in ad hoc networks, such as remaining battery power, location, and speed. One critical feature of ANMP and any AHN network management protocol is its ability to handle the dynamic nature of the nodes in the network as normal events and not exceptions. This includes nodes dying, moving, joining the network, and belonging to multiple networks.

ANMP utilizes a three-level hierarchical architecture, depicted in Fig. 3.1. The top level is the network manager and the bottom level consists of the nodes in the network, called agents. Several agents close to each other are grouped together to form clusters and each cluster has a cluster head. These cluster heads, which are managed by the network manager, manage the agents and form the middle level of the hierarchy. ANMP also includes a user interface, making management more user-friendly and effective.

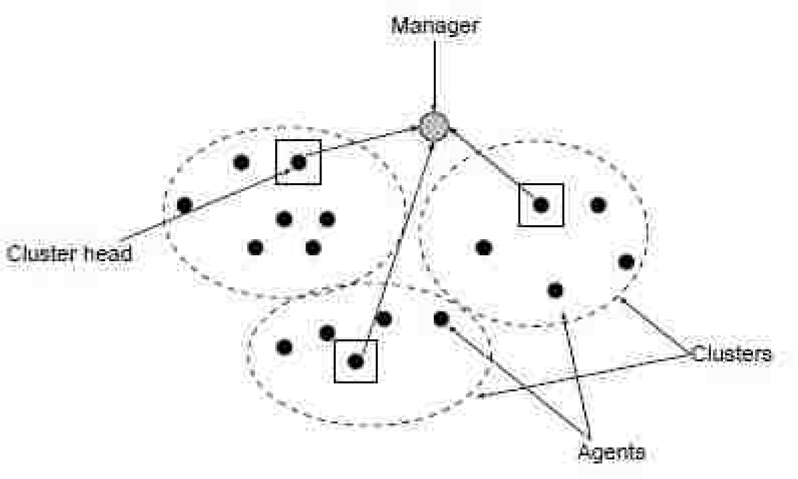


Figure 3.1: Hierarchical architecture of ANMP [27].

Despite research performed in the area of network management of wireless sensor networks, a standard has not emerged. One system that has been developed is MANNA [28]. It is different from most network management systems in that it “considers three management dimensions: functional areas, management levels, and WSN functionalities”, instead of the two (functional areas and management levels) defined in traditional network management. Fig 3.2 illustrates the relationship among these three dimensions, which are all considered when defining a management function. MANNA also comprises three sub-architectures: functional, information and physical. The functional architecture defines how management functionalities are distributed among manager, agents and management information base. This distribution can be centralized, distributed or hierarchical. How this functional architecture is implemented is the physical architecture of MANNA. The information architecture is object-oriented and consists of classes representing the resources under the three management dimensions.

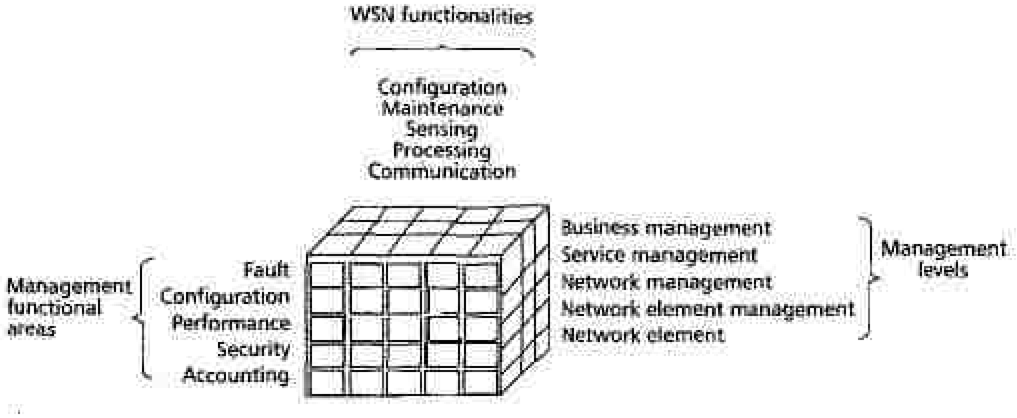


Figure 3.2: MANNA management functionality abstractions [28].

Another WSN management system is the Sensor Network Management System (SNMS) [29]. SNMS provides two management functions. One function is event logging which is event-driven. This feature allows nodes to report their data if they meet conditions specified by the user. The other management function collects data from the nodes, both physical characteristics, such as remaining battery power, and sensed data, such as temperature. Besides only having limited functionality, SNMS also monitors in the passive mode only, in response to a user query. This monitoring imposes little network bandwidth or processing overhead.

3.1.1 Network Management Systems with Ontology

The utilization of ontology in network domains has seen extensive research in recent years. There has not been a NMS developed for a HMN; however, some related work has contributed to the network management domain. Table 3.1 identifies the related works in this section and provides an overview of their primary features.

Table 3.1: Comparison of NMS.

Work	Wired Tier	AHN Tier	WSN Tier	Goal
Wong [30]				Automatically map management concepts
López de Vergara [31, 32, 33, 34]				Common management model
Cleary [35]			√	Configuration management
Moraes [36]	√			Performance management
Orwat [37]		√		Security management
OntoSensor [38, 39]			√	Trend discovery in sensor measurements
New HMNMS	√	√	√	Network management (topology discovery tested)

One area of research related to the work here is the interoperability support provided by ontology and the mapping of various network management concepts. As the mapping techniques advance and become stable, they may be incorporated into the development of the mapping ontology used in the HMNMS, which was manually created.

According to Wong, et al [30], interoperable systems must share in data or knowledge exchange, exhibit coordinated behavior, and cooperate in problem solving. They proposed a “method of automatic ontology mapping based on a semantic similarity function” [30]. This was accomplished by developing a concept similarity estimation and an ontology mapping. The network management concepts researched were represented in First Order Predicate Calculus (FOPC). The degree of similarity between the FOPC statements was measured and then an ontology mapping procedure was developed. The first step in the ontology mapping procedure was a scheme that classifies matching results according to their FOPC similarity values. The classification scheme is then used to guide the search process through a target ontology, which is the ontology traversal algorithm.

López de Vergara, et al [31, 32, 33, 34] present how an ontology can assist in the comparison of different management information languages, including the semantic expressiveness of these languages. Their research has concentrated on semantically integrating management information from different network management models, such as SNMP and CMIP. This was done by obtaining behavior characteristics through the use of rules, axioms and constraints, which are all parts of an ontology. Fig 3.3 illustrates the mapping process. Management specifications from different management models are merged into one ontology using semantic mapping rules. This mapping will lead to the different management models being able to understand the semantics of the other models.

There has been research in the use of ontology for various aspects of network management. The majority of this work is for one-tier networks, not HMNs, in a specific area of management. The following works highlight some of the applications of using ontology in specific areas of network management.

Configuration management is one area of network management that may take advantage of ontologies. This is because of the large amount of human interaction necessary for configuration tasks and the similarities between configuration management and other problems related to knowledge sharing, reuse and reasoning. In [35], Cleary,

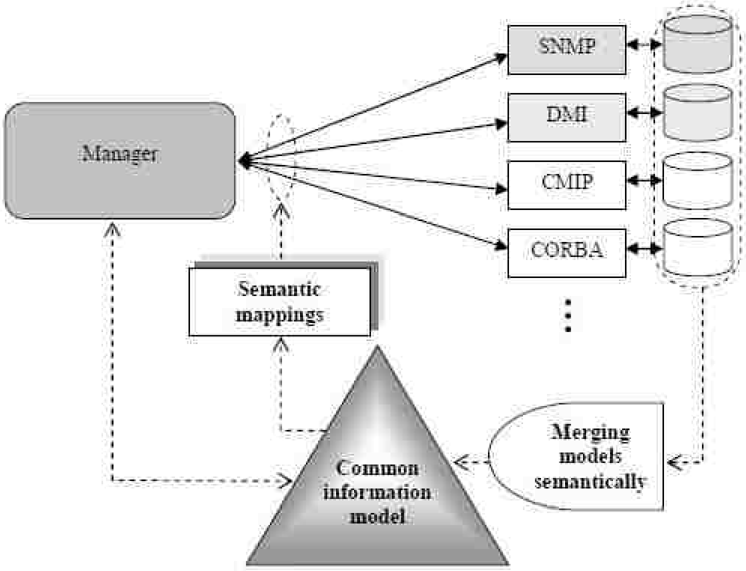


Figure 3.3: Merge and map process for network management information [33].

Danev and O’Donoghue developed a new network modeling approach that is based on ontology. They applied this approach to wireless networks. The new application interacts with a traditional network management system via an XML representation of the configuration data. The overall architecture for this approach is shown in Fig 3.4. WCDMA-RAN is the Radio Access Network (RAN) used, which uses the WCDMA (Wideband Code Division Multiple Access) communication protocol [40]. The application reads the XML data and uses it to create ontology instances.

These instances are supplied to the inference engine. The engine suggests possible configurations and also validates the consistency and integrity of user configurations against the knowledge base. The new configuration is converted to XML and fed back to the NMS for deployment to the network. The engine uses three different types of expert rules to validate existing configurations and suggest possible configurations for use. The new approach reduces the amount of human interaction needed for configuration tasks.

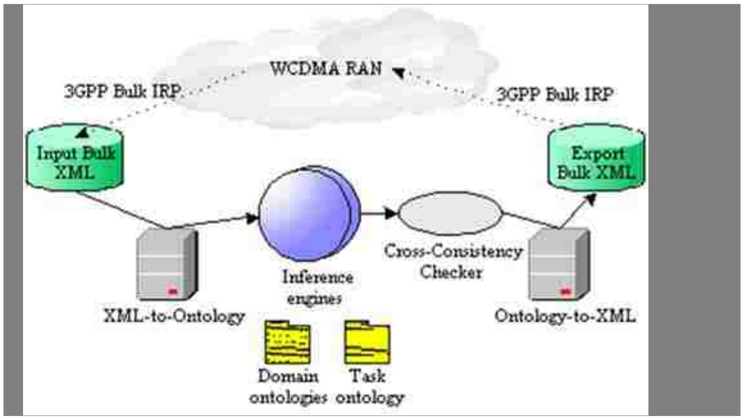


Figure 3.4: Ontology centric architecture [35].

Moraes, Sampaio, Monteiro, and Portnoi [36] developed an ontology, MonONTO, that can be used primarily in performance management, including quality of service and monitoring. MonONTO was used with an expert system that could determine application performance based on previous performance. The previous performance is learned from the network and fed to the knowledge base. The knowledge base contained ontology instances about advanced network applications, application users, and network monitoring. These instances were used to determine the most likely network performance in a given situation. This work was for wired networks in determining application performance in a specific network environment.

Ontologies have also been researched to assist in providing a secure management environment for Mobile Ad hoc Networks (MANETs) [37]. Orwat et al. created MANET Distributed Functions Ontology (MDFO), which was “used to structure MANET performance and security information” [37]. This new approach provides a mechanism to assist in making decisions for dynamic configuration changes in MANETs. It will also provide a foundation to incorporate security factors to enhance the decision processes in

MANETs. One part of MDFO is the translator, which will convert information collected from the network into ontology semantics. The output of the translator will populate a database with static and dynamic information about MANET devices. The database is queried when a MANET function is necessary and will then create instances in the ontology and send relevant attribute values to the decision making process. MDFO can "serve as the basis for MANET decision making and optimization and correspondingly both control and facilitate the conduct of MANET operations" [37]. This research is the first step to providing optimized management of MANET functions and services.

OntoSensor [38, 39] is a domain ontology designed for a heterogeneous sensor network prototype environment. OntoSensor extends the upper-level IEEE Suggested Upper Merged Ontology (SUMO) [41], which defines general concepts and associations. It is also built on SensorML [42], which is a generic data model that defines associations and properties common to sensors. The base station includes an OntoSensor ontology. Information about the sensors, including the data acquisition boards, sensing elements, and processors, is included in the repository. The repository responds to ad hoc queries to assist in trend discovery in the measurements. The prototype environment only covers devices in the 2006 Crossbow [43] catalog and requires a priori knowledge of the platform class of each sensor.

3.2 Analytical Models for Network Performance Analysis

There has been some research on the development of an analytical model for performance analysis of heterogeneous networks. A comparison of these models is summarized in Table 3.2. The primary difference of the new analytical model developed in this research is that it is for conducting a performance analysis on a multi-tier network. Prior work was done on models for single-tier networks.

Table 3.2: Comparison of Analytical Models.

Work	Homogenous or Heterogeneous Network	Number of Tiers in Network	Queuing Model Used
Ismail and Zin [44]	Heterogeneous	Single-tier	M/M/1
Hedayati, Kamali, and Izadi [45]	Heterogeneous	Single-tier	M/M/1
New Analytical Model	Heterogeneous	Multi-tier	M/M/1

Ismail and Zin [44] developed a simulation model based on queuing theory. The model was developed to be used to measure the performance behaviors of a live network. The model was developed to analyze the performance of a heterogeneous environment over a Wide Area Network (WAN), specifically in an institution of Higher Education.

The model developed was based on the M/M/1 queuing theory model. It was developed by studying a heterogeneous environment in a live network. The information learned from the heterogeneous environment was converted into a logical model.

The live heterogeneous environment was at a Higher Educational Institution. It consisted of a Local Area Network (LAN) at a main campus and a WAN connecting a branch campus. The goal was to develop a model to study the performance of services over the WAN connection.

The model was used to find the total size of various packet services of all the clients in the heterogeneous environment, Trafik_Heter. The model was:

$$Traffic_{Delay} = \frac{\mu_{Jumlah}}{\frac{J_{LAN}}{v} + \left(\frac{\mu_{Jumlah}}{C_{LAN}}(n+1)\right)} + \frac{\mu_{Jumlah}}{\frac{J_{WAN}}{v} + \left(\frac{\mu_{Jumlah}}{C_{WAN}}(n+1)\right)} \qquad (3.1)$$

where μ_{Jumlah} is the total size of packet services requests by clients, J_{LAN} is the LAN distance, J_{WAN} is the WAN distance, v is the speed of light, C_{LAN} is the LAN bandwidth, C_{WAN} is the WAN bandwidth, and n is the total nodes in the two networks (LAN and WAN).

Services were run in a live network environment. Remote data transfers were simulated in the live environment and the propagation and transmission delays were measured. Results from the simulation model were less than one second, typically within tens of milliseconds, to the actual values. This confirmed that the simulation model can be used to estimate data transfer times in a heterogeneous environment over a LAN and WAN.

There were several assumptions made in the simulation model. These assumptions were that there was no packet loss, no jitter in delays and sufficient network bandwidth. Jitter refers to the difference in the end-to-end delay (arrival times) among packets. While the model was used in a heterogeneous network environment, it was for a single-tier network.

A similar approach to network traffic monitoring was proposed by Hedayati, Kamali, and Izadi [45]. Similar to the previous approach, Hedayati, et al proposed a model based on M/M/1 queuing theory. The model was developed to simulate and monitor the network traffic of a heterogeneous LAN environment.

The live network used to verify the model was a university LAN. The results of the live network tests were similar to the simulation model. This confirmed that the simulation model can be used to calculate network throughput (the rate for successful delivery of messages on the network, often in some form of bits per second) and congestion rates (the amount of data on a network that causes delays in packet delivery) for a live heterogeneous network.

The model developed was for calculating the instantaneous congestion rate, $A_0(t)$, and the stable congestion rate, A_C. The equation developed for the instantaneous congestion rate was

$$A_0(t) = P_1(t) = \frac{1}{(m+1)(1-e^{-(m+1)t})} \qquad (3.2)$$

where $P_1(t)$ is the arrival probability of the queue length for the router's group at time t and m is the service rate. The following equation was developed to calculate the stable congestion rate

$$A_C = P_1(C+1) = 1 - m/(((1+m+C_p w)A_0(C-1)) - (1+(C-1)m(1-A_0(C-1)))A_0(C-1)+m)) \qquad (3.3)$$

where C is the routers' buffers.

As with the previous work, this analytical model was developed as a simulation model for a heterogeneous network. It is for a single-tier network, not for multi-tier networks.

3.3 Intrusion Detection Systems

3.3.1 Basic Intrusion Detection Systems

Huang and Wicks [46] use the analogy of an intrusion to that of a battlefield. In intrusion detection as well as in the battlefield they cite a number of shared characteristics including an environment that is heterogeneous and widely distributed, a significant amount of data that is constantly changing and which can be extremely noisy, incomplete and inconclusive information that makes decision making difficult, and attack patterns which are constantly changing. One must take these characteristics into account when devising mechanisms for intrusion detection.

Huang and Wicks point out that if a file-access-violation is detected, the true purpose of this event cannot be determined without additional information referred to as context. Such contextual information would include such information as the present machine configuration, the location of the files, permissions, and account configuration. The important point that Huang and Wick make is that by the time sufficient information arrives at a central analysis point, the situation (context) may have changed drastically. Huang and Wicks' approach to analyzing what may be happening is to consider the strategy the attacker may be using. This in turn calls for a description of the attacks that are more abstract in nature. This is consistent with the approach described in this research, namely to represent descriptions of attacks in the form of a conceptual ontology.

In Camtepe and Yener [47] an approach to detecting complex attacks is presented that is based on the construction of finite automatons that represent the "patterns" of complex attacks. They define a non-deterministic enhanced finite automata to be a tuple consisting of Q, a set of states, Q_{PA}, a set of partial attack states, Q_A, a set of attack states, F, the input alphabet, D, a set of derivation rules for goals and subgoals, and $DELTA_F$ and $DELTA_B$, sets of forward and backward transition rules. The finite automata can recognize complex attack patterns. The automata implicitly specifies the relationships between the attack elements and therefore, unlike a conceptual representation, possesses no ability to generalize or specialize exists without the specification of another automaton.

A Process Queuing System (PQS) was the method used in [48] to detect complex attacks. The complex attacks were represented as finite state machines (FSM) with the attack elements represented as states and the transitions were triggered by observations about the occurrence of an attack element or a response to an attack element. The FSM were represented as models, which could be incorporated into a hierarchy of models, allowing for high-level models to be developed to detect complex attacks based on results of lower-level models.

A system was developed, PQSNet, to demonstrate the application of PQS to network security. PQSNet utilized existing sensors, such as Snort [49], firewalls, system log files, etc. to obtain security information. Information from the sensors were fed into a PQS model. FSMs were used in PQSNet to represent complex attacks with each step in a complex attack represented as a finite state. When an alert is received from a sensor indicating an event occurred, a transition will occur. A general sample of a FSM in PQSNet is depicted in Fig. 3.5. The Start state indicates that there has been no malicious activity, the Recon state indicates that some reconnaissance activity was detected, and the Attacked state indicates that the host was attacked. PQS supports model tiering, which allows the output of one model to become input to a higher-level model. This characteristic allows PQSNet to abstract basic attacks, which permits complex attacks to be written in an easier manner in higher-level models.

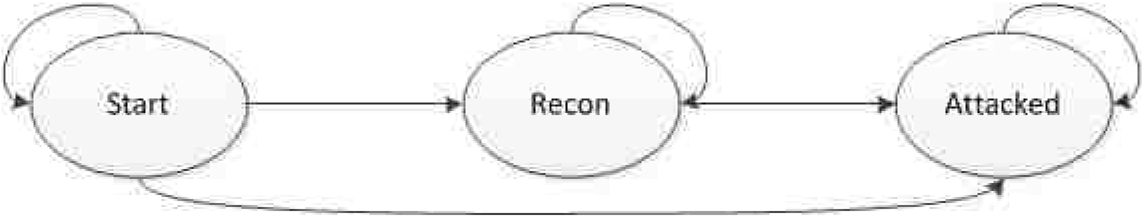


Figure 3.5: General FSM in PQSNet [48].

Snort [49, 50] is a common, open-source, network-based IDS. Snort is primarily a signature-based IDS, with its signatures called rules. The rules in Snort contain sufficient expressive power to detect simple attacks. Detecting complex attacks, which consist of multiple packets, is more complicated and requires cross-event analysis in Snort. This task requires preprocessors, which are more resource intensive than rules. Anomaly-based detection is also possible with some of Snort's preprocessors.

The Snort architecture, shown in Fig. 3.6, consists of several components. The traffic on the network is captured using a packet sniffer. The packets captured are then sent to any configured preprocessors. The detection engine is responsible for applying the rules to the captured packets looking for matches, which results in alerts. The alerts are written to files or a database for viewing by the network manager.

One feature of Snort is its configurability. This adds some complexity but also much flexibility, as it allows each administrator to configure Snort for their network deployment and use, as well as their IDS needs. There are many configuration options

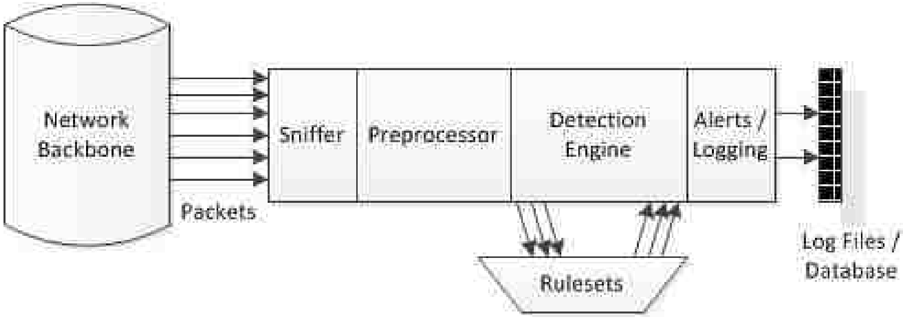


Figure 3.6: Snort architecture [50].

available in Snort, including the choice of which rules to incorporate. Another option is the addition of selected preprocessors, which will do some processing prior to rule processing on the data. Some of the possible preprocessors are protocol checks for common protocols, packet re-assembly for fragmented datagrams and port scanning. The preprocessors allow for more complex intrusion detection.

Snort is a real-time IDS, meaning it will run the preprocessors and rules against network packets as the packets pass through the Snort engine. For this reason, Snort may not process all packets because of the speed of the network and the amount of data passing through its engine. In order to behave in real-time, Snort will skip some packets and not process them. This will allow some network attacks to get through the Snort implementation and into the network.

The rules provide the ability to configure Snort to meet a network’s needs and quickly adapt to new attacks. The rules also lead to a disadvantage in Snort. The addition of new rules to handle new attacks has led to a rapid growth of the rule set in Snort (see Fig. 3.7). This requires more time to process packets and perform the pattern matching against the rules. This will lead to performance degradation and fewer packets processed by Snort, which will result in more false negatives.

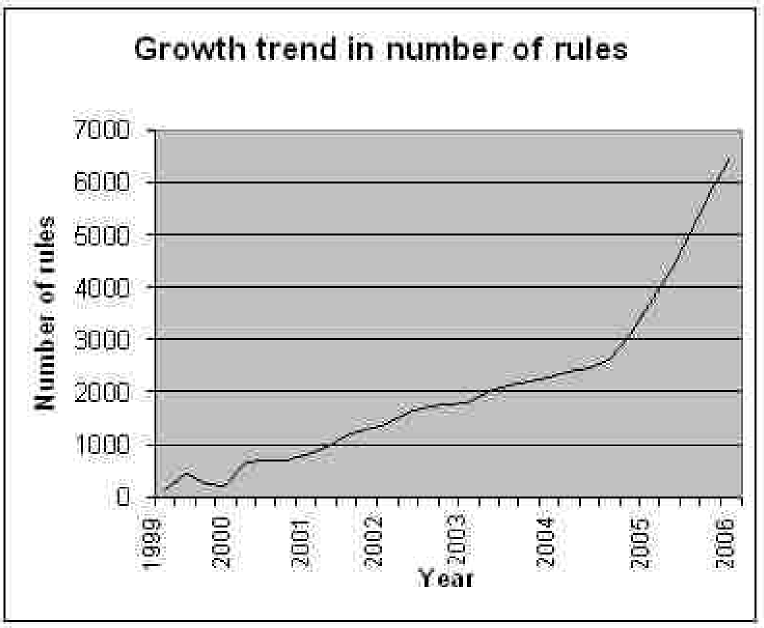


Figure 3.7: The growth trend of the number of rules in Snort [51].

Although Snort is an IDS and will generate alerts for attacks detected, there is still considerable manual analysis required. The recommended manual analysis when using Snort [50] is to first check the priority of the alert generated. Any low priority alerts, which indicated an important alert in Snort, are further analyzed. All alerts involving any critical device on the network, which must be identified by the organization, are identified and investigated. Any source address appearing in multiple alerts is further investigated. Well-known attack methods, such as using static source ports and IP fragments, are identified. If these attack methods target a weakness in the network, this should be addressed by incorporating a security measure to strengthen the weakness. As time permits, which it often does not, the network manager prioritizes the remaining alerts and further examine the one prioritized high.

3.3.2 Reasoning-based Intrusion Detection Systems

Various RIDS research is presented here. Each uses some type of logical reasoning in the IDS. Table 3.3 presents a high-level comparison of the research presented. The table indicates if the IDS detected attacks against hosts or the network, if the IDS detected complex attacks, and how ontology was utilized in the IDS. The IDS research presented in this dissertation is denoted as “new IDS”.

Table 3.3: Comparison of RIDS.

Work	Host or Network	Detects Complex Attacks	Ontology Use	Goal
MulVAL [52]	Host	√	No ontology, used Datalog	Vulnerability analysis
Xu, et. al. [53]	Both		Common vocabulary for security information	Formal representation of alert analysis
Martimiano, et. al. [54, 55]			Common vocabulary for security tools	Model concepts for security incidents
Tsoumas, et. al. [56]			Common vocabulary for security requirements	Security management system based on interoperability, aggregation and reasoning
ReD [57]	Both		Instantiate new security policies after attack detection	Detect and react to attacks
Vorobievf, et. al. [58, 59, 60]	Both	√	Common vocabulary for IDS components	Detect attacks using common vocabulary among distributed components
Undercoffer, et. al. [61, 62]	Host	√	Model computer attacks	Detect attacks
Mandujano, et. al. [63, 64]	Network	√	Represent attack signatures and environment characteristics	Detect outgoing intrusions
New IDS	Both	√	Represent and detect attacks	Detect complex attacks based on traffic data

The MulVAL [52] system uses a logical deduction process to determine the existence of a multistage attack on a network. It is a framework to model the interaction between software bugs and the configurations of nodes on the network (systems and network devices). This framework, illustrated in Fig 3.8, consists of generic rules, including rules to determine if a vulnerability exists and the consequence of an exploit against the vulnerability. MulVAL is used to filter attack information and only output essential data for the system administrator to analyze.

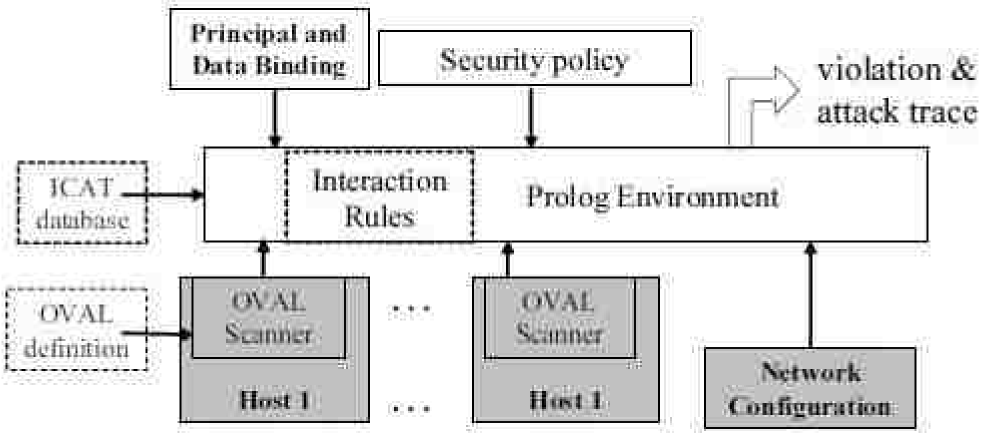


Figure 3.8: MulVAL framework [52].

There are six different inputs to MulVAL's analysis. The first input is the advisories. These consist of the vulnerabilities, which are then checked for existence on each machine. This is done by using an OVAL (Open Vulnerability Assessment Language) [65] scanner. The results of this scanning process are converted to Datalog clauses. Datalog [66] is a query and rule language that is a subset of Prolog. To understand the effect of each vulnerability NIST's National Vulnerability Database (NVD, formally ICAT) is used, with the relevant information also converted to Datalog clauses.

The configurations for each host and network device to be scanned are two more inputs to MulVAL. Host configuration information includes the software and services running on the host as well as their configurations. The OVAL scanner is also used to gather host configuration information, with the output once again converted to Datalog clauses. The network devices that are a part of MulVAL's analysis are limited to routers and firewalls. These configurations are manually created using Datalog clauses.

Information about the principals, or users of the network, is another input to MulVAL. These Datalog clauses map a principal to its accounts on the various network hosts. Additional Datalog clauses describe the policies of the network, which indicate the data access for each principal.

The last input is a model of how all the components interact. These interactions, represented as Horn clauses, include a pattern that can be matched to identify a multistage attack. Instead of coding specific vulnerabilities for the interactions, the vulnerabilities were generalized, preventing frequent rule changes.

The OVAL scanner is run on each host with the output reported to the host running MulVAL. The scanner must be run on each host and identifies vulnerabilities specific to each host. MulVAL will then run an analyzer on the properties received from all the scans. This analysis is done in two phases, an attack simulation phase and a policy checking phase. The attack simulation phase identifies all possible data accesses of an attacker, which are then sent to the policy checking phase. This phase will compare the output of the attack simulation phase with the specified security policy and identify violations. Both of these phases utilize a Datalog program, but the separation of the phases provides for the possibility of using a richer policy language for the policy checking phase without affecting the complexity of the attack simulation phase.

An important feature of MulVAL is the ability to reason about multistage attacks. This is done through the use of Horn clauses that are created for the semantics of the vulnerability and the operating system allowing the determination of an adversary's options in each stage of a multistage attack. The use of generalizations of attack methodologies in the interaction rules allows the rules to be more static; however, since MulVAL uses vulnerability recognition in its scanning process, a scanner must be run on each host to be monitored. Also, when a new vulnerability report is utilized, each host must be re-scanned. The authors of MulVAL concentrated their efforts on denial of service and privilege escalation attacks only.

3.3.3 Reasoning-based Intrusion Detection Systems with Ontology

Context-aware alert analysis was researched by Xu, Xiao, and Wu [53]. They argue that alert analysis for unified security management can be divided into three stages: alert collection, alert evaluation, and alert correlation. An ontology was developed following a four-step process: 1) model the conceptual level, 2) define the model in OWL [15], 3) define correlation rules using SWRL [67], and 4) define security management services using OWL-S. OWL-S can be used to provide a semantic description to Web services. The overall architecture is shown in Fig 3.9.

The ontology developed was based on the CIM (Common Information Model) Schema [68]. The key concepts include context, asset owner, vulnerability, threat and countermeasure. The context was used for alert evaluation. Alert correlation was achieved by extending the ontology with SWRL, which adds behavior information through the use of rules. As an example, if a host is running an FTP service and a specific operating system, then the attacker may be able to learn operating system information about that host. Attack scenarios were built from the defined SWRL rules. OWL-S was used to define security management policies for automatic response.

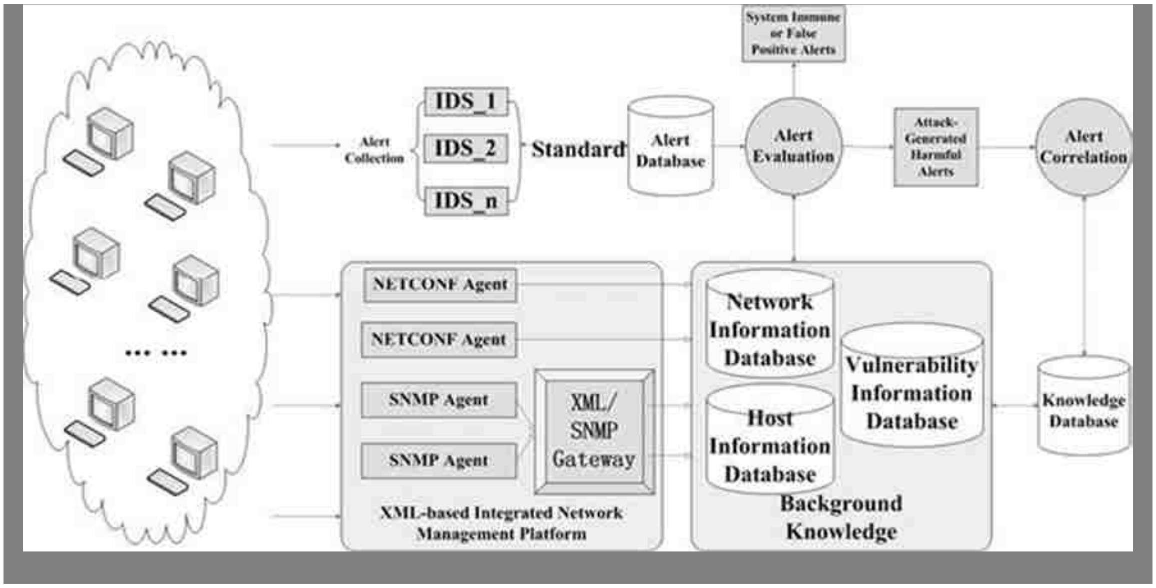

System Immune or False Positive Alerts
Alert Collection
IDS_1
IDS_2
IDS_n
Standard
Alert Database
Alert Evaluation
Attack-Generated Harmful Alerts
Alert Correlation
NETCONF Agent
NETCONF Agent
SNMP Agent
SNMP Agent
XML/ SNMP Gateway
XML-based Integrated Network Management Platform
Network Information Database
Host Information Database
Vulnerability Information Database
Background Knowledge
Knowledge Database
... ...

Figure 3.9: Proposed architecture for context-aware alert analysis [53].

The system proposed takes input from multiple IDSs, both HIDS and NIDS, and integrates it into the knowledge base. Reasoning rules are used to perform alert correlation and build attack scenarios.

The work of Xu, et. al. is similar to the work described in this research as they both examine attack scenarios, but focused on attacks against Web Services, while the system to be described in this research focuses on all types of attacks on any node on the network. The context-aware alert analysis was the foundation of an ontology to provide security knowledge in a uniform manner, available to multiple security tools or systems, although it is not used for the identification of multi-phased, complex attacks.

Martimiano and Moreira [54, 55] focused their research on what they identified to be the difficult problem in security management: “efficiently generate knowledge about security to make decisions and solve security incidents”. One of the problems with solving security incidents is that the various security tools often used by system and network administrators generate data in different formats. The authors developed an ontology called ONTOSEC to assist with solving security incidents. The main concept was the Security Incident class and all other classes related to this class. The other primary classes included access, agent, asset, attack, consequence, time, tool, and vulnerability. The main concepts and relations for ONTOSEC are shown in Fig. 3.10.

Attacks identified by this system assume that all security incidents exploit a vulnerability. The information in the vulnerability ontology was based on the CVE (Common Vulnerabilities and Exposures) project [69] and NIST’s NVD. Attack information was obtained from Snort [49] rules. The primary attributes used to identify a security incident were the source IP address, destination IP address, security incident type, date, time, weekday, description, reference, and severity.

ONTOSEC was validated using a data driven approach by comparing it with the source data about the domain. The source data used for comparison was Snort alerts. The ontology developed was used to provide a common format to be shared by various security tools. It will store security incident data but not identify security attacks. A security incident can precede and/or succeed another incident but there is no mention of identifying multi-phased, complex attacks.

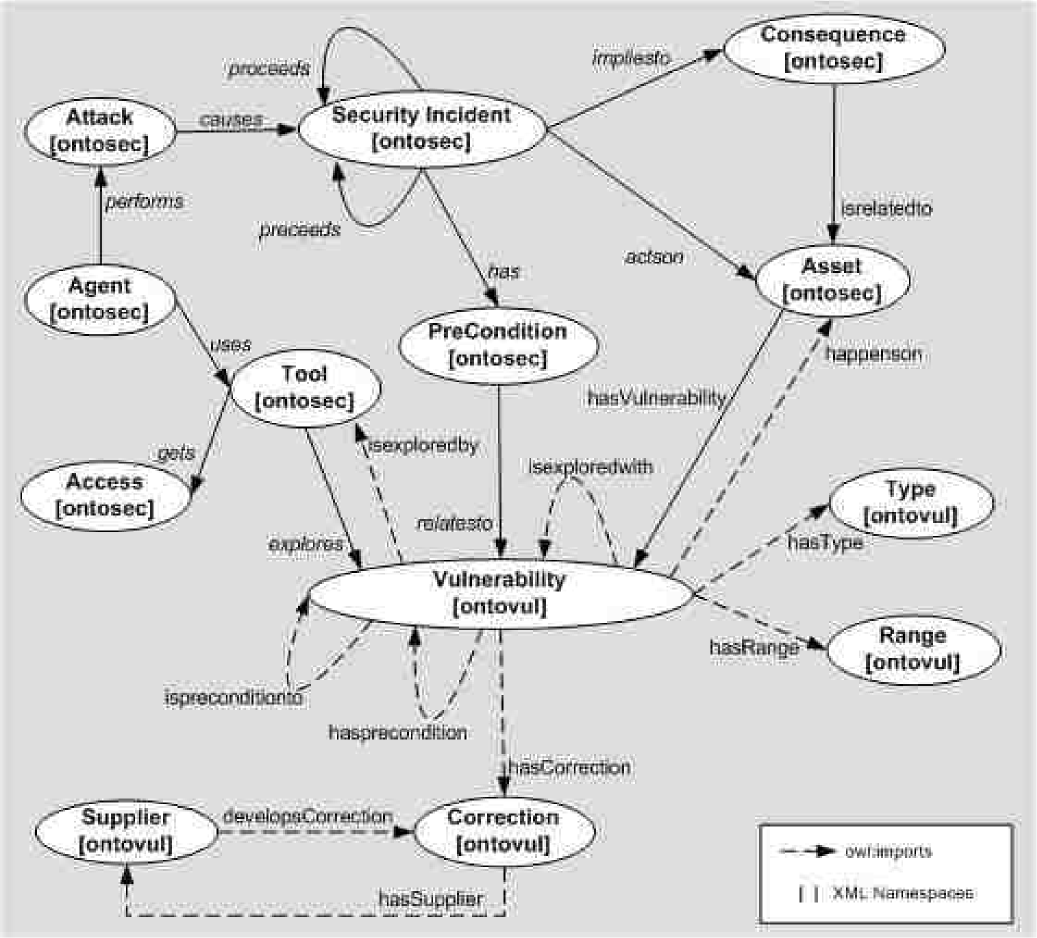


Figure 3.10: Main concepts and relations in ONTOSEC [54].

Tsoumas and Gritzalis [56] developed a “knowledge-based, ontology-centric security management system” used to “bridge information system (IS) risk assessment and organizational security policies with security management”. They extended the CIM (Common Information Model) standard to create a generic Security Ontology (SO). The development consisted of a model of the conceptual level, which was an extension of CIM and then implemented in OWL. Their work consisted of four phases, building an ontology, collection of security requirements, definition of security actions, and deployment and monitoring of the system security.

The first phase was to build the Security Ontology. This included the use of scanning tools to get data from the assets in the infrastructure being monitored. The organization’s mangers were consulted to discuss business decisions made about the security environment. From the infrastructure data retrieved from the assets, instances were created in the ontology.

Security requirements were collected in phase two; security knowledge was extracted from the IS policy document and used to create ontology instances. The security requirements were evaluated by management and security experts for correctness.

Phase three consisted of defining security actions. The security requirements were associated with specific security controls. These controls were then transformed into a form that could be used for Ponder rules. Ponder is a language used to specify security policies in a common way.

The fourth and final phase is the deployment and monitoring of security actions. The Ponder rules that were created in phase three were deployed in the IS infrastructure. The last important step in the process was to iterate from step one again, in a timely manner. This was necessary to continually iterate over the steps to keep current with the changes in the IS environment and policies.

Their work focused on security requirements in a centrally managed location. It abstracted security requirements by extending the CIM Schema into OWL ontologies. The ontology developed was focused on risk assessment and demonstrated that security information can be extracted from risk assessment countermeasures.

Various tools were used to get infrastructure data, such as the network topology, servers, active ports, etc. The security management requirements, including information from security policies, were entered into the knowledge base manually and were then linked to security controls for countermeasure identification.

These ontologies were used for knowledge sharing and to provide risk assessment support. The system they developed did not utilize an IDS or identify security attacks. The primary goal was to combine risk assessment and an organization's security policies to assist with security management.

The ReD (Reaction after Detection) project [57] defined and designed solutions to enhance the detection and reaction process of network attacks. A framework was developed to find the best way to react to a network attack, both for the short- and long-term. The architecture, shown in Fig. 3.11, consisted of five components: 1) the Policy Instantiation Engine (PIE), 2) the Alert Correlation Engine (ACE), 3) the Policy Decision Point (PDP), 4) the Reaction Decision Point (RDP), and 5) the Policy/Reaction Enforcement Point (PEP/REP).

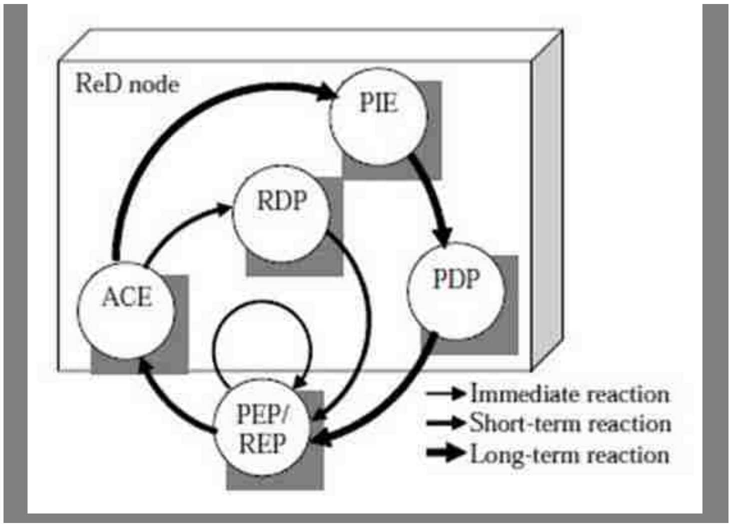


Figure 3.11: ReD architecture [57].

The proposed ontology was used to instantiate new security policies in reaction to identified attacks. The alerts and policies were defined in the ontologies and inference rules were used to map the alerts into attack contexts. The architecture also utilized the Detection Message Exchange Format (IDMEF) [70] for exchanging alerts among elements and OrBAC (Organization Based Access Control) [71] as the policy language.

Alerts were sent from the network nodes to the ACE, which performed some analysis to detect an attack. The ACE sent the attack information to the PIE, which instantiated new security policies to react to the attack. The new policies were sent to the PDP, which deployed the policies to the PEP/REP for enforcement. The RDP also received information about the attacks from the ACE and determined mid-level reactions to the attack.

Three types of reactions were defined, low-, mid-, and high-level. These classifications were based on the level of diagnosis that was required to apply the reactions. Low-level reactions were decided by the PEP/REP and immediately enforced. The RDP decided on mid-level reactions based on attack information it received from the ACE. These reactions did not include new security policies. The PIE determined the high-level reactions, which resulted in the generation of new security policies that were eventually deployed.

The PIE was the center of the architecture. It mapped the IDMEF alert information in the Alert Ontology and the OrBAC reaction policy in the OrBAC Ontology. The PIE used these ontologies, along with the alert information received from the ACE to determine which components required a reaction and what that reaction should be.

SWRL rules were used to infer the hierarchy information in the OrBAC model, map IDMEF alerts to OrBAC holds, and obtain the necessary security policy.

The mapping of the attack alerts information to security policies was their focus. The policy instantiation process is shown in Fig. 3.12. The ontologies developed were used to identify and instantiate the security policies necessary to react to an attack; they were not used to detect an attack. The attacks were detected by using modified Snort IPSs, syslog daemons, and host-based IDSs.

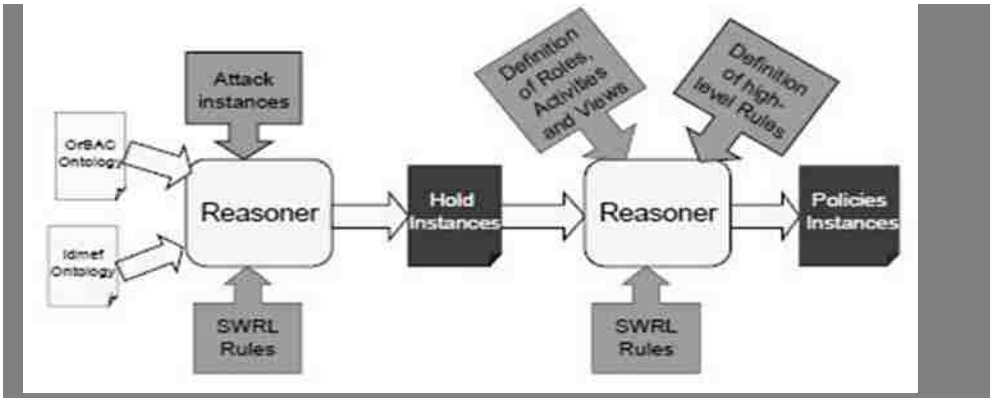


Figure 3.12: Policy instantiation with ontologies. [57].

Reasoning was a part of its architecture. It was used to infer the mapping from IDMEF alerts to OrBAC policies. The ontologies received alerts from the IDSs (NIDS and HIDS) and syslogs. From these alerts, analysis could be performed about attacks, including multi-phases, complex attacks. The focus was on the mapping from security attacks to security policies.

Vorobiev, Han, and Bekmamedova [58, 59, 60] discussed how distributed firewalls and IDSs (F/IDSs), monitoring different hosts, must work together in a distributed manner. They evaluated five different types of attacks: attacks against Web Services, P2P attacks, Denial of Service attacks, sniffing attacks, and multi-phased, distributed attacks. Their research paid particular attention to the gaming industry and the implementation of gaming systems using the component-based software system (CBSS) and peer-to-peer (P2P) approaches.

A framework was developed that used a variety of components from different vendors that acted as a coalition. The primary component was called a defensive component (DC).

The research also resulted in the development of several ontologies. The Security Asset-Vulnerability Ontology (SAVO) was the main ontology in the system and gave a simplified view of information security. It was the high-level ontology and included classes to describe the various aspects of the system, including attack, vulnerability, defense, risk, and threat agent. The ontologies were developed to assist in simplifying security information. The Security Attack Ontology (SAO) and the Security Function Ontology (SFO) were both used by the system to provide a common vocabulary to the other ontologies. The SAO defines the classes for specific types of attacks, such as a Web Services attack or a Peer-2-Peer attack. The defenses against each of these attacks are defined in the Security Defence Ontology (SDO). The SFO was used by developers to define protections against security attacks and failures. The Security Algorithm-Standard Ontology (SASO) was used to define security algorithms and standards used in the system.

As part of the framework, shown in Fig 3.13, Snort instances were deployed throughout the network. These Snort instances sent information about attacks to the DCs.

When a DC detected a new attack, it added the attack to the SAO, which was then shared with the other members of the coalition. If a coalition member developed a defense against a new attack, a countermeasure was added to the SDO, which was distributed to all coalition members. The manager, which was running the framework engine, decided how to react to the attack and sent orders to the DCs for action.

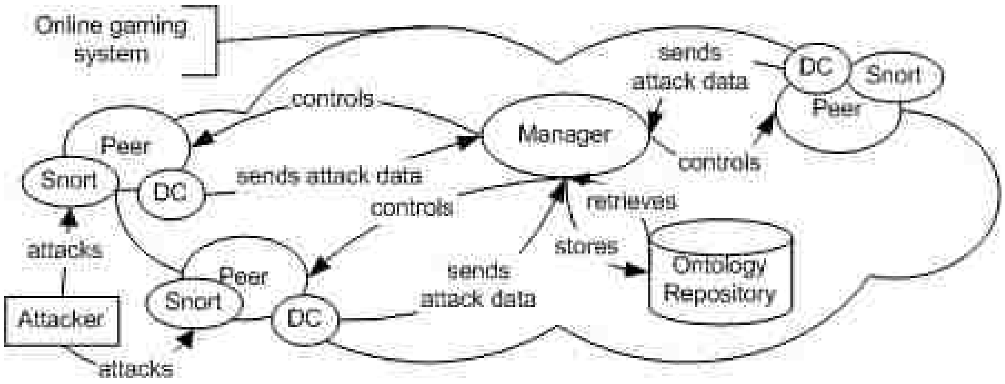


Figure 3.13: Prototype implementation [60].

The ontologies in this framework provided a common vocabulary for the distributed F/IDSs. These worked collaboratively to detect multi-phased, complex attacks. When a host identifies an attack, it shares this information with the other hosts in the framework, which then use the shared information to detect a multi-phased, complex attack. Each host is required to implement a F/IDS, where the IDS portion is an HIDS. The framework also includes countermeasures against identified attacks.

Undercoffer, Joshi and Pinkston [61, 62] produced work that performed analysis to identify various elements of an attack, including the means or method, consequence, target, and most common origin location. “An intrusion is comprised of some input resulting in some consequence, while the impact is directed towards a system component, received from some location and causes some means of by inducing some system behavior" [62]. The target of the attack refers to the specific component of the target and could be classified as the network layers, kernel-space, application, or other. The method used by the attacker, the means, was categorized as an input validation vulnerability, a general exploit, or a mis-configuration of the target or one of its components. The consequence refers to the end result of the attack and may be one or more of denial of service, the attacker achieves user access to the target system, the attacker achieves root access to the target system, there is a loss of confidentiality, or some other undesired result. The location refers to the origin of the attack in relation to the target of the attack. Possible values for location are remote (another network), local (same network), or either local or remote (may be either on another network or the same network). The high-level overview of these concepts in the ontology can be seen in Fig. 3.14.

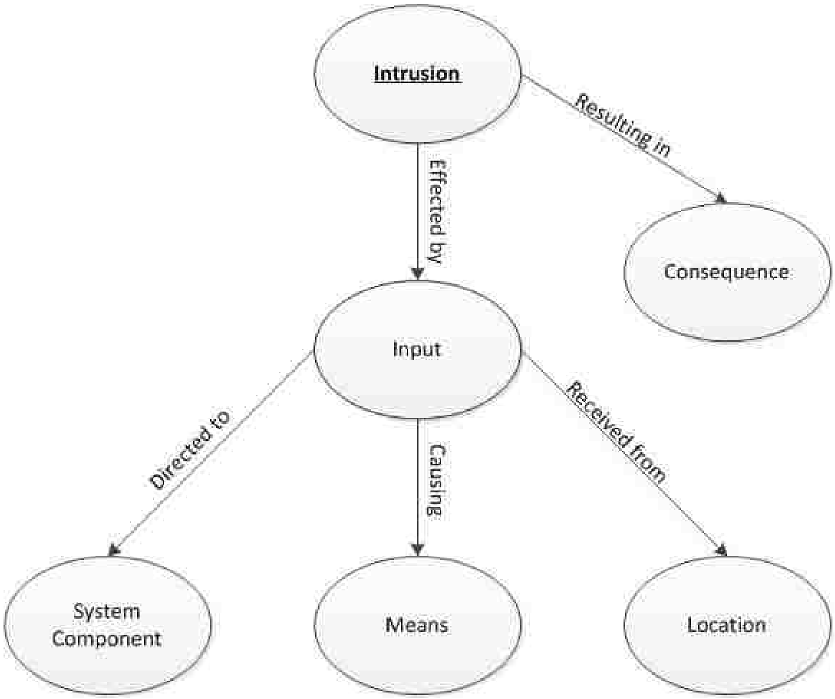


Figure 3.14: High level overview of ontology [62].

Vulnerability information was taken from CERT/CC advisories and the National Vulnerability Database (NVD) [72], formally the Internet Catalog of Assailable Technologies (ICAT), maintained by NIST (National Institute of Standards and Technology). The host attributes, such as network connections, memory usage, open connections, etc. were monitored and the state of the host was determined. This was done by an IDS monitoring the host, requiring an IDS to be installed on or adjacent to all hosts that are a security concern.

The authors developed a taxonomy based on these attack elements. The taxonomy was defined in terms of observable relationships and measurable characteristics of the target, such as the total memory, average CPU load, instruction pointer value, and number of child processes running. The ontology was developed from the taxonomy and centered on the target of the attack. The system learned normal behavior and then used the ontology to detect anomalies in the behavior.

This research utilizes IDSs that send security alerts to the ontology, which will then infer information about attacks. The system performs reasoning to detect multi-phased, complex attacks. One host detects a simple attack that is one step in the multi-phased, complex attack, while another host may detect another step of the multi-phased, complex attack. This information is combined so the multi-phased, complex attack occurrence can be inferred. It focused on hosts as the targets of all attacks, which is not always the case when dealing with network security. To do damage to more aspects of the network, an attacker may target a network device, thus attempting to take down an entire subnet or network. The research described in this work focuses on any node as the target of an attack, including hosts and network devices, such as switches, routers, and firewalls.

An ontology-based intrusion detection system was described by Mandujano [63] and Mandujano, Galvin, and Nolazco [64]. In this approach, the authors are looking to detect outgoing intrusions using a multiagent system. A multiagent system utilizes multiple software agents to gather data for input into the system. The goal of an OID is to help protect remote systems. This work accomplished OID by taking advantage of the fact that many complex attacks are automated using scripts or executable programs. The system developed analyzed changes in the network traffic and the resources used by an automated attack tool. The resources were identified by evaluating the program profile during execution. The agents were used to collect data, detect possible incidents, and implement reactions to the incidents identified.

The ontology developed for the system was an attacker-based ontology, focusing on the originating user or system. The ontology identified all elements about the system, including automated attack tools, network traffic, signatures, sensors, and reactions, as well as their relationships. The ontology they propose enables the detection of code and network activity that identifies a possible intruder. The ontology specifies concepts like hostile and safe processes as subclasses of a process, for example. Their ontology, unlike the ontology proposed in this research, does not distinguish between traffic and attack. It is our contention that such a distinction is necessary to successfully identify sequences of incoming attacks and also to be able to recognize the type and kind of attack that is transpiring.

Much of the research has concentrated on attacks against hosts. Only the ReD Project utilized NIDSs for alert information, the other research used HIDS or no IDS. This research will detect attacks against any node on the network, including network devices such as switches, routers and firewalls. Network devices can provide attackers with very valuable information about the network and hosts. For instance, if an attacker identifies a password for a network device, it may also be a password used on other network devices or perhaps even servers on the network. A significant consideration is that a compromised network device is much less likely to be detected.

Much of the previous work is focused on identifying vulnerabilities of systems and evaluating the threats against these targets. This work will focus on the network traffic and not the vulnerabilities or targets. By doing this, it is possible to identify attacks and also attack attempts, even if the vulnerability doesn’t exist in the target node or network. This may be the result of the target of the vulnerability not being deployed in the network, or the target may have been patched to resist the vulnerability, etc. It is important to note that attack attempts are just as important or meaningful as an actual attack. The attempts can alert the administrator to an attacker existing that is trying to penetrate their network or a node on their network. It also allows the administrator to prepare the future deployments such as a user adding a web server to the network that may contain vulnerabilities.

This work will begin with specific attack examples but will evolve into more general cases. The rules developed for identifying complex, multi-phase attacks will be generic, and will lead to the identification of any type of attack, including zero-day attacks. These rules will allow a family of complex, multi-phased attacks to be defined and detected. By representing these attacks ontologically, a more advanced and reusable representation of network attacks will be created.

Chapter 4 **Network Management of a Heterogeneous, Multi-tier Network**

4.1 Challenges of Heterogeneous Multi-tier Network Management

When considering a HMN, each tier in the network may have its own network management system or protocol; there may even be varying management systems or protocols within one tier. This presents a considerable problem for proper network management; it is difficult to exchange information between disparate systems. This requires the network manager to gather information from several management stations.

Network management software exists that can help manage a heterogeneous network. An example is ProIT [73] by PerformanceIT, Inc. This software utilizes SNMP to retrieve data from devices from a variety of manufacturers. The software is primarily used for performance and fault management. Configuration management, a common network management function, is often too manufacturer-dependent for third-party software.

Another disadvantage of third-party management software solutions is the need to install add-ons or agent software for the management station to retrieve the data from the devices. Typically software must be installed for each different manufacturer. As new devices are added to the network, often an upgrade must be done for that manufacturer’s add-on to allow proper communication with the new devices.

Each new manufacturer and device must be configured in the management software. This is often a device-by-device task, but some software does allow some group configurations for similar devices. This configuration is a time-consuming process.

Another issue is that the majority of network management systems gather raw measurement data only. There is no semantic information gathered. For example, the NMS may state that there were 567 dropped packets on interface 23, but that data alone is meaningless. The semantic information for this might specify that 567 dropped packets may be a critical concern since it is over a specified threshold; however, semantic information about that particular interface indicates it is a printer that may have a higher threshold for dropped packets so there is no alert generated. In existing NMSs, the analysis to create the semantic information must be carried out by the user – in this case the network manager. Even if one system was able to gather semantics about the network, there is no way for the various systems to exchange semantic information because there is no standard way to represent this semantic information.

Four domains of a network system have been identified, all requiring management. The four domains are Nortel wired, Cisco wired, ad hoc, and wireless sensors. The challenge is to bring coherence to a network system that consists of different types of equipment, described in different ways, to provide a unified view to a NMS. An investigation transpired to see if it would be possible to create a unified NMS that would be usable for each of the identified types of networks while at the same time providing a common view of these networks. In addition to these requirements for the solution, it must also be scalable and adaptable.

4.2 An Overview of an Ontology-based Network Management System

We investigate a potential solution that meets these requirements, specifically a unified approach to network management. This approach uses an ontological representation of the various networks rich enough to express raw and semantic information and at the same time able to be processed by appropriate algorithms.

The ontological representation provides the backbone of the integration of disparate systems. From a computational approach, an ontological representation can be algorithmically acted upon thus allowing us to apply processing power to determine what is going on in any of the network types. Most deployed systems are able to provide a network manager with information of the state of the network passively. The correct knowledge represented ontologically can be acted upon using expert knowledge to provide a richer description of the state of the network. We hope to create an NMS that operates at a level significantly exceeding those that exist today by employing this knowledge.

The new design for an NMS using these ideas [74] is shown in Figure 4.1. The system contains a Graphical User Interface (GUI), an Ontology Subsystem, an Ontology Instances Interface, and descriptions of the network management protocols for each network type (wired, AHN, and WSN). Each component will be explained.

The Ontology Subsystem consists of three components: the ontology, the knowledge base, and the reasoner. The ontology is explained in the next section. The knowledge base contains the ontology definition files and raw instances of all devices deployed in the HMN. When the NMS is launched, the ontology definition files are loaded into the knowledge base. Also during the NMS launch, instances are added to the knowledge base for all active deployed devices. The instances are added by the Ontology Instances Interface, which is explained soon.

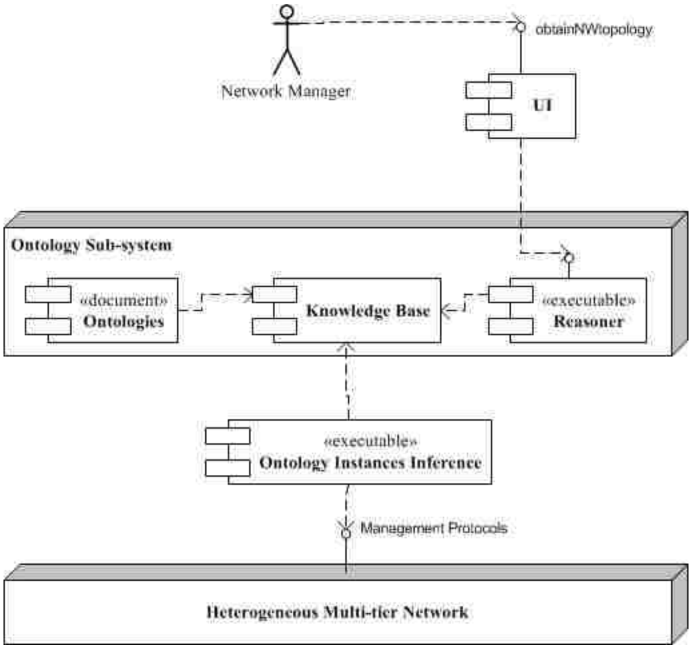


Figure 4.1: Component diagram of the Network Management System (NMS) [74].

The reasoner is the part of the NMS that allows a network manager to interact with the knowledge base. The FaCT++ [75] reasoner is used in this research. FaCT++ is a description logic (DL) reasoner, which provides logical reasoning for ontologies. The GUI obtains queries from the network manager and then interfaces with the reasoner to obtain the query answer from the knowledge base, which returns the results to the GUI. The GUI then displays the results to the network manager.

The Ontology Instances Interface (OII) is a program that interfaces between the nodes in the HMN and the knowledge base, which contains the ontology definition files and data instances for deployed devices. The OII periodically sends a management query to each node in the network. The query depends on the network type and the management protocol used for that network type.

For instance, for wired devices the query is an SNMP query. When the node receives the query, it will extract the raw data that it has been maintaining that answers the query and create a response. Upon receiving a response to the query, the OII will extract the raw data from the response packet and create an instance in the knowledge base for that node. For example, if the query asks node *Node1* for its name, location and description, then *Node1* retrieves that information from its memory, creates a response packet containing this information, and sends the response back to the OII. The OII will then extract this information (the name, location and description for *Node1*) from the response packet and create an instance in the knowledge base for node1. This instance will contain the information returned by the node (its name, location and description).

The management query is sent to the deployed devices by utilizing existing network management protocols, when possible. The wired devices are queried using SNMP. When the wired node receives an SNMP query, it will retrieve the MIB values and return them to the OII. The OII will create an instance from the MIB values returned and add the instance to the knowledge base. A separate query is sent for each wired node deployed in the network.

SNMP is also used for ad hoc devices. For this to happen, a new MIB and ad hoc agent were created for ad hoc networks [76]. The new MIB contains properties for retrieving battery information, such as the battery life remaining, both in percent and seconds, and if the battery life is low. The basic properties for ad hoc nodes, such as name, location, serial number, IP address, etc. are retrieved using the new ad hoc agent but using existing SNMP MIBs.

WSN sensors are statically defined. In the future, management protocols or systems will be utilized to obtain the device information for this network type as well. There are no current standards for network management protocols for WSNs, so the protocol or system used to obtain the data for this network type will be determined based on its maturity and effectiveness within the newly developed NMS. For the sensors in the WSN, the sensor Network Management Protocol (sNMP) [77, 78] and the Sensor Network Management System (SNMS) [26] are two options for use to send the management query. When incorporated into the NMS, the data will be obtained from sNMP or SNMS, just as it was done with SNMP, and the device instances will be added to the knowledge base.

4.3 The Ontology-based Approach

The domain of the ontology developed is HMNs. The ontology forms the basis for our approach to managing such networks. The ontology will answer questions about all tiers and devices of the network. Examples of questions that can be answered by the ontology are:

- Where is each device located?
- What is the address of all devices?
- What is the energy level of the device?

These questions were used as a starting point for the definition of the ontology domain.

The first step in ontology development [79, 80] is to define the terms for the domain. Terms are the vocabulary of the domain or the things that need defined or explained to the user. Some of the terms for the ontology in the network management domain are: name, location, address, energy level or residual energy, node role (cluster head or member node) and status. After all the terms are defined, the following steps are followed to construct the ontology [81]:

1. The classes and class hierarchy are defined

2. The class properties or slots are defined

3. The facets of the slots are defined

4. Instances of the classes are created

The ontologies are written using OWL as the knowledge representation language. OWL was chosen because of the expressiveness required for the HMNMS. The primary expressiveness necessary in the ontology that is provided by OWL and not provided by other knowledge representation languages are the specification of disjoint classes and the mapping of common terms. It is necessary to specify that some classes are disjoint to gather more semantic information about the deployed devices. For instance, if a device is characterized as a Nortel device by being a member of the *Nortel* class, then the device cannot be a Cisco device since the *Nortel* and *Cisco* classes are specified as disjoint. The mapping of common terms means that terms in multiple domains with the same meaning can be mapped to one common term in a mapping ontology file. For example, the serial number for a device is maintained in both the Nortel and Cisco devices. In the Nortel domain, the term used for the serial number is *rcChasSerialNumber* and in the Cisco domain the term is *chassisSerialNumber*. In order to have the required interoperability for one NMS, these two terms must be represented by one common term in the ontology.

The classes representing the various device types, class hierarchy, class properties and facets are defined in the ontology. The class hierarchy consists of the various types of devices that may be deployed in a HMN (see Fig. 4.2). The main class is the *Node* class, which contains properties that exist in any network node, such as name, description, and serial number. The wired and wireless nodes are subclasses of the *Node* class and contain properties that exist in each of these network domains. Currently the *Wired* class has no additional properties; the *Wireless* class has a role (cluster head vs. non-cluster head node), a status, and the remaining energy. The subclasses of each of these two classes will be the various types of wired and wireless nodes. Currently, wired nodes consist of Nortel devices and Cisco devices and the wireless nodes are either ad hoc nodes or sensors. Seven ontology definition files were developed, for simplicity, corresponding to the nodes in the class hierarchy. The complete OWL code for the Network Management System is provided in Appendix A.

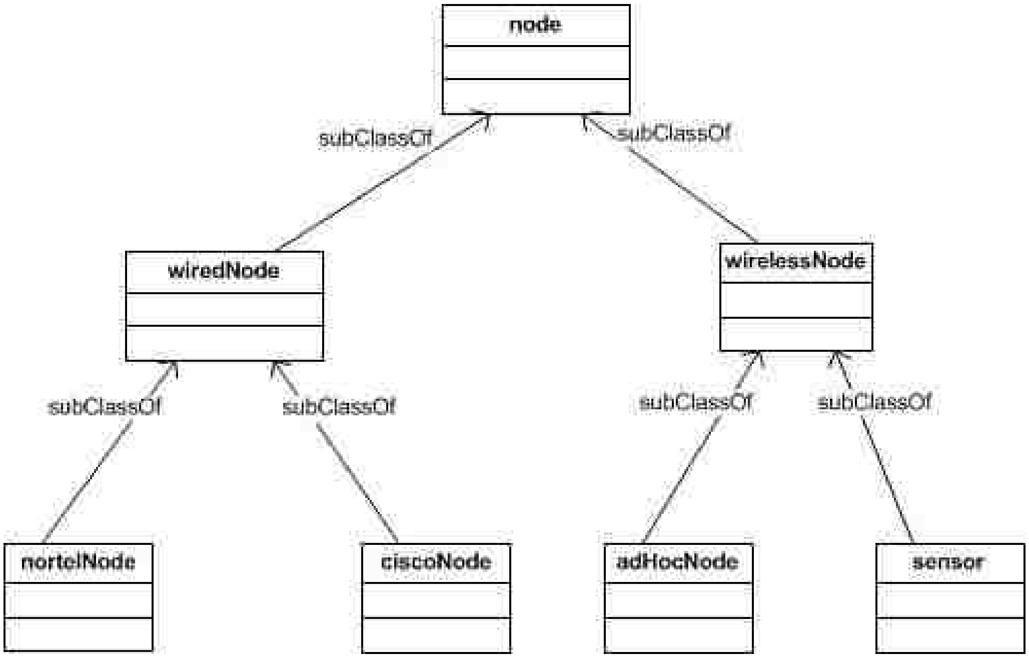


Figure 4.2: Class hierarchy for the HMNMS ontology.

The Nortel and Cisco classes are quite similar. Each one contains the same fields, such as the IP address, subnet mask, system description, system name, and chassis serial number. The reason for two different classes is because of the proprietary nature of the manufacturer’s MIBs. For instance, Nortel and Cisco use different terms for the chassis serial number, as previously discussed.

The fields in the AHN and WSN classes are similar and represent items such as the node address, location, serial number, remaining energy, role, cluster head, and status. The role and cluster head fields are used in clustering to identify if the node is a cluster head or an agent/member node and to identify an agent/member node’s cluster head. Two different terms are used to correspond to common technology in each network type (agent for AHN and member node for WSN). The status field indicates if the node is active or inactive (also alive or dead in the case of a WSN node).

In order to deploy an ontology application for network management, the data must be mapped to one domain, using a mapping ontology. This ontology definition maps data from the four main classes (*Nortel*, *Cisco*, *Ad hoc*, and *Wireless Sensor*) into one class by taking similar data from each network type and mapping it into a common term (see Table 4.1). The development of the mapping ontology definition requires domain knowledge and interpretation of this knowledge. Comprehensive ontologies, developed by domain experts, reduce the burden on network managers. For example, as discussed in the previous section, Nortel and Cisco each use a different MIB identifier for the chassis’s serial number. This requires these two fields to be mapped to a common term, *serialNumber*.

As another example, consider the network device's address. Wired and AHN devices may use an IP address but a WSN node may use an IP address or simply a node ID (1, 2, 3, etc.). This research used a node ID for the address of WSN nodes. In order to list all network device address's in a HMN, the IP address in the Nortel, Cisco and Ad hoc ontologies are mapped to an address field and the node ID in the WSN ontology is mapped to the same address field. This allows a network manager to ask once for a list of all devices and their addresses (IP or node ID). Without the mapping ontology, the network manager would have to query four different NMSs separately to get all deployed devices with their corresponding addresses.

Table 4.1: Common Terms in the Ontology.

	Cisco Domain	Nortel Domain	Ad hoc Domain	WSN Domain	Common Term
System Name	*sysName*	*sysName*	*name*	*name*	*name*
System Location	*sysLocation*	*sysLocation*	*location*	*(xcoord, ycoord)*	*location*
System Description	*sysDesc*	*sysDesc*	*description*	*description*	*description*
Serial Number	*chassisSerial Number*	*rcChasSerial Number*	*serialNumber*	*serialNumber*	*serialNumber*
Address	*sysIPAddr*	*rcSysIPAddr*	*ipAddress*	*nodeID*	*address*
Subnet Mask	*sysNetMask*	*sysNetMask*	*subnetMask*	N/A	*subnetMask*
Role (cluster head or member node)	N/A	N/A	*role*	*role*	*role*
Status (alive/active or dead/inactive)	N/A	N/A	*status*	*status*	*status*
Remaining Energy	N/A	N/A	*remainingBat teryLife*	*residualEner gy*	*energyLeft*

The energy left in AHN and WSN nodes, which is a primary concern because of the limited energy resources, is another use of the mapping ontology to assist the network manager. If the network manager wants to know how much remaining energy is in each

AHN and WSN node, the manager would have to query multiple sources, possibly even each individual node if no NMS was implemented for these two network types, which is often the case. By utilizing the ontologies developed in this research, the manager could ask one query, which consults the mapping ontology and return the remaining energy of all AHN and WSN nodes. This allows the network manager to easily find all nodes that have energy levels of concern for further evaluation.

4.4 Implementation of the Ontology-based Network Management System

One facet of configuration management is topology discovery. Topology discovery for a HMN is, at best, a difficult task. Other aspects of configuration management that may benefit from a new NMS are determining the current status of deployed devices, knowing when a configuration needs to be updated, and determining the future deployment status of devices. In particular, topology management can answer questions posed by the network manager regarding the current, and potentially the future, status of deployed devices.

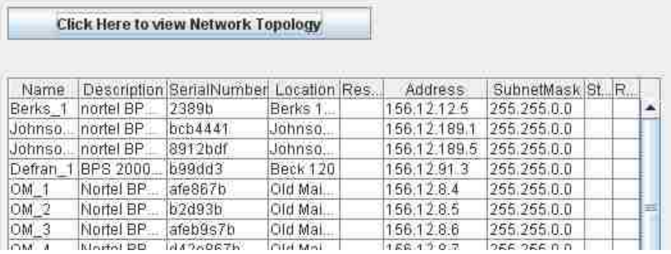
Network topology is one network management task that is important to all network managers. It is important for a network manager to know the devices that are deployed and some properties for each. Also, many configuration management tasks rely on the network topology. For these reasons, obtaining the network topology was the task that was the focus of the prototype system for this research and performed on each test network implementation.

4.4.1 An Experimental Heterogeneous Multi-tier Network

An ontologically-based NMS was deployed on a Windows machine for testing purposes. The tests were run with a heterogeneous three-tiered network that was created for evaluation of the new NMS. The HMN network consisted of various numbers of wired (Cisco and Nortel), ad hoc, and sensor nodes. The wired portion of the network was simulated using a node emulator. The emulator was an implementation of the SNMP agent that would exist in deployed wired devices. The emulator responded to the SNMP requests with SNMP responses corresponding to unique wired nodes. The responses from the nodes were captured by the Ontology Instances Interface component of the NMS, which created instances in the knowledge base for each node. This portion of the network behaved in the same fashion as a live wired network and allowed testing of all aspects of the NMS. To study the performance and correctness of the developed ontology the network also contained AHN and WSN nodes. The AHN and WSN nodes were statically defined in the ontology and directly loaded into the knowledge base when the NMS was initiated.

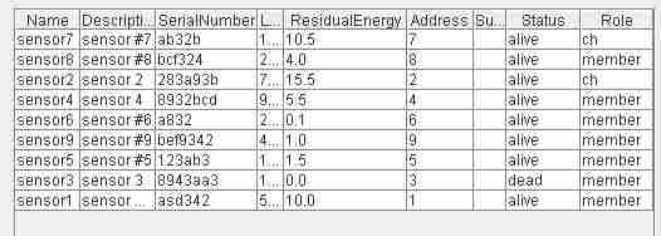
Each test network was deployed, with a different number of nodes, in a simulation environment, instances were created for each device deployed, and the network topology was obtained and displayed. Ten trials of this experiment were run for each network implementation with the arithmetic mean used for comparison purposes. The percentage of wired nodes (70%), AHN nodes (10%), and WSN nodes (20%) was the same for each experiment.

A topology discovery was performed on the test HMN using the new NMS. The NMS correctly retrieved basic properties from each deployed device in the HMN using the data stored in the knowledge base. Fig. 4.3 and Fig. 4.4 show the GUI snapshots for some of the wired devices and WSN devices respectively. These snapshots show an example of the results returned when the network manager asks to see all properties for all deployed devices in the network. As demonstrated by the figures, the properties returned vary for device type, specifically between wired and wireless nodes, since the properties are different for the different device types.



Name	Description	SerialNumber	Location	Res...	Address	SubnetMask	St...	R...
Berks_1	nortel BP...	2389b	Berks 1...		156.12.12.5	255.255.0.0		
Johnso...	nortel BP...	bcb4441	Johnso...		156.12.189.1	255.255.0.0		
Johnso...	nortel BP...	8912bdf	Johnso...		156.12.189.5	255.255.0.0		
Defran_1	BPS 2000...	b99dd3	Beck 120		156.12.91.3	255.255.0.0		
OM_1	Nortel BP...	afe867b	Old Mai...		156.12.8.4	255.255.0.0		
OM_2	Nortel BP...	b2d93b	Old Mai...		156.12.8.5	255.255.0.0		
OM_3	Nortel BP...	afeb9s7b	Old Mai...		156.12.8.6	255.255.0.0		
OM_4	Nortel BP...	d42e867b	Old Mai...		156.12.8.7	255.255.0.0		

Figure 4.3: Characteristics of several wired devices in the HMN [74].



Name	Descripti...	SerialNumber	L...	ResidualEnergy	Address	Su...	Status	Role
sensor7	sensor #7	ab32b	1...	10.5	7		alive	ch
sensor8	sensor #8	bcf324	2...	4.0	8		alive	member
sensor2	sensor 2	283a93b	7...	15.5	2		alive	ch
sensor4	sensor 4	8932bcd	9...	5.5	4		alive	member
sensor6	sensor #6	a832	2...	0.1	6		alive	member
sensor9	sensor #9	bef9342	4...	1.0	9		alive	member
sensor5	sensor #5	123ab3	1...	1.5	5		alive	member
sensor3	sensor 3	8943aa3	1...	0.0	3		dead	member
sensor1	sensor ...	asd342	5...	10.0	1		alive	member

Figure 4.4: Characteristics of several WSN devices in the HMN [74].

The results of the query for all properties of all deployed devices in the simulated network are shown in Fig. 4.5 (KB – knowledge base). When the number of devices is relatively small (less than 100), the overhead of the HMNMS (time to add instances to and retrieve query results from the knowledge base) is less than half of the total time. The total time is the time to initialize the HMNMS (add all deployed devices to the knowledge base) and retrieve the network topology.

As the number of deployed devices grows over 100 devices, the query time increases to more than half of the total time. This is due to the scalability of the knowledge base. As the network grows, the scalability is an issue. For most network managers, the scalability issue will be an acceptable trade-off as the total time is still less than the time that is required when a manual collection of data is necessary for HMNs. The scalability of ontology is addressed as future work and is discussed in chapter 8.

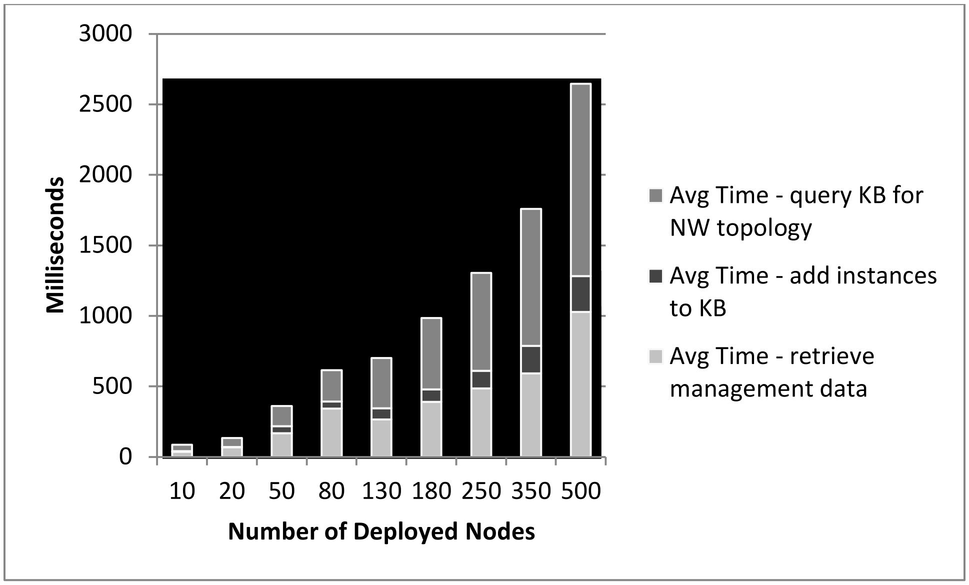


Figure 4.5: Results for the HMNMS for a simulated network.

4.4.2 A Test Heterogeneous Two-tier Network

The test network for this deployment of the HMNMS consisted of two of the three possible tiers, wired and ad hoc. Sensor nodes were not part of this test network because of the lack of a standard management protocol. The wired tier consisted of both Cisco and Nortel nodes.

The wired nodes were previously configured with SNMP data, which was part of the standard installation of the network devices. This included information such as the IP address, network mask, name, location, etc. There was no additional configuration required for their deployment to the test network.

The ad hoc nodes were laptops running Linux with an ad hoc routing protocol installed, which is necessary for a multi-hop ad hoc network (an AHN with multiple connections between a node and the gateway). The newly-developed SNMP ad hoc agent and MIB were installed, which required some additional installation and set-up.

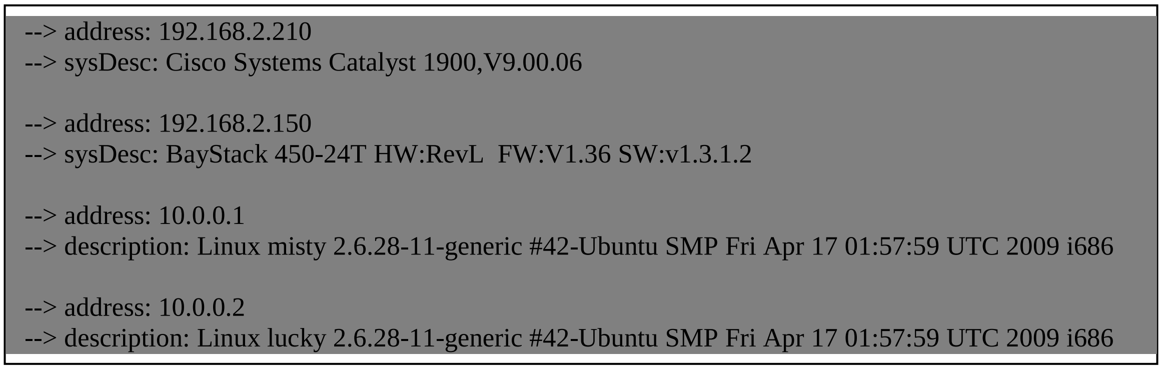
The IETF developed the Agent Extensibility (AgentX) Protocol [82] to dynamically extend SNMP agents. The AgentX protocol splits the agent into two separate parts, a master agent and subagents. The master agent is a traditional SNMP agent but has no access to management information on the nodes. The subagents have no SNMP knowledge but have access to the management information on the nodes. The subagents communicate to the management station via the master agent. The management station sends the SNMP queries to the master agent, which then communicates the required information to the subagents by using the AgentX protocol. The subagents retrieve the requested information from the node's memory and return it to the master agent, which then sends it to the management station.

The newly-developed ad hoc agent was created as an AgentX subagent, so an AgentX master agent is also required. The AgentX master agent used in this research is the Net-SNMP distribution [83]. When the laptops are booted, the AgentX master agent and new ad hoc AgentX subagent are started and ready to answer SNMP requests.

The HMNMS is run on a management station that is part of the test network. The management station is able to access the wired network, the ad hoc network, and the developed ontology definition files. The ontology definition files are stored on a web server so the management station can read these files via the Internet. The management station is used to query the ontological knowledge base, and the results of the NMS analysis of the data being collected.

A simple query is sent from the network manager, via the GUI, to the knowledge base requesting the address and description of all deployed nodes in the test HMN. When the HMNMS is initiated, all deployed devices are queried and the responses are added to the knowledge base via the OII. At that point, the knowledge base contains the ontology definition files and instances for all deployed devices. So, when the GUI sends a query to the knowledge base, it is sent via the reasoner. The reasoner will send the query to the knowledge base and retrieve the answer for the query. The query answer is then returned to the GUI where the network manager views it. The address and description of all active deployed devices is returned to the network manager in response to the query because of the interoperability provided by the incorporation of ontology in the HMNMS.

A portion of the query results is shown in Fig. 4.6. These results show the information requested (address and description) for four of the nodes in the network. The first two nodes are wired nodes (the first one is a Cisco node and the second one is a Nortel node. The last two nodes are ad hoc nodes deployed in the network. As seen in the results, both ad hoc nodes are hosts running Ubuntu versions of Linux.



```
--> address: 192.168.2.210
--> sysDesc: Cisco Systems Catalyst 1900,V9.00.06

--> address: 192.168.2.150
--> sysDesc: BayStack 450-24T HW:RevL  FW:V1.36 SW:v1.3.1.2

--> address: 10.0.0.1
--> description: Linux misty 2.6.28-11-generic #42-Ubuntu SMP Fri Apr 17 01:57:59 UTC 2009 i686

--> address: 10.0.0.2
--> description: Linux lucky 2.6.28-11-generic #42-Ubuntu SMP Fri Apr 17 01:57:59 UTC 2009 i686
```

Figure 4.6: Sample query results from the HMNMS for a test network.

4.4.3 Deployments in Live Networks

In consideration of the contributions of this work, the system was deployed in live environments to obtain a quantitative measurement of the performance and deployment of this solution. The HMNMS uses existing management protocols to obtain node information. This contributes to the ease of deployment for the HMNMS. Deployed network nodes will most likely already support the standard management protocol by default. If a node does not support the standard management protocol, it is easily enabled by changing a configuration setting in the device. Deployed nodes require no additional software to be installed to work with the HMNMS.

The HMNMS system is installed on a single management station. This requires the installation of the FaCT++ reasoner and the Ontology Instances Interface. The reasoner requires access to the ontology definition files; they can be copied onto the management station or onto a web server that is accessible to the management station.

The HMNMS was deployed and tested in two live network environments. The first was a corporate network consisting of Cisco devices and the second was a university network with Nortel devices. While the two live networks were homogeneous deployments, these deployments provided an opportunity to test the ease of wide-scale deployment of the HMNMS.

As with the test network, the HMNMS deployment was a minor issue and required no configuration changes to the network devices. The deployed devices were all configured for SNMP and required no additional software or firmware installation.

The network management station was a laptop that was connected via Ethernet to the network. The laptop was running the FaCT++ reasoner to handle the ontology knowledge base. The HMNMS was already compiled on the laptop so the only requirement was to run the HMNMS utilizing the specified ontology files. In the current deployment, the devices must be manually characterized as Cisco or Nortel. After the completion of that manual step, the HMNMS properly gathered the necessary information from the deployed devices.

Results of these live deployments demonstrated that the ontology sub-system overhead was minimal. A comparison of the performance results for the university network is illustrated in Fig. 4.7 (KB – knowledge base; props - properties). These results illustrate that the majority of the response time is the query for the SNMP data, which exists in all NMSs utilizing SNMP and it not unique to the HMNMS. As noted in the figure, instances are added to the knowledge base swiftly. In this live network, which is a realistic view of the actual utilization of the HMNMS, the query response time for the network topology is minimal. This response time grows as the number of devices grows, but it is acceptable provided the benefits provided.

The time to add instances to the knowledge base for all deployed devices in the network is relatively constant, even as the number of deployed devices increases, as shown in Fig. 4.8. The figure also shows that the time to retrieve the network topology for all deployed devices in the network grows quickly as the number of devices grows. This increase in query time does not increase as quickly as the time to retrieve the management data from the deployed devices. The time to retrieve management data is present in any NMS and the query time is still less than the time to conduct manual analysis for a HMN, so the growth is acceptable provided the benefits of the HMNMS.

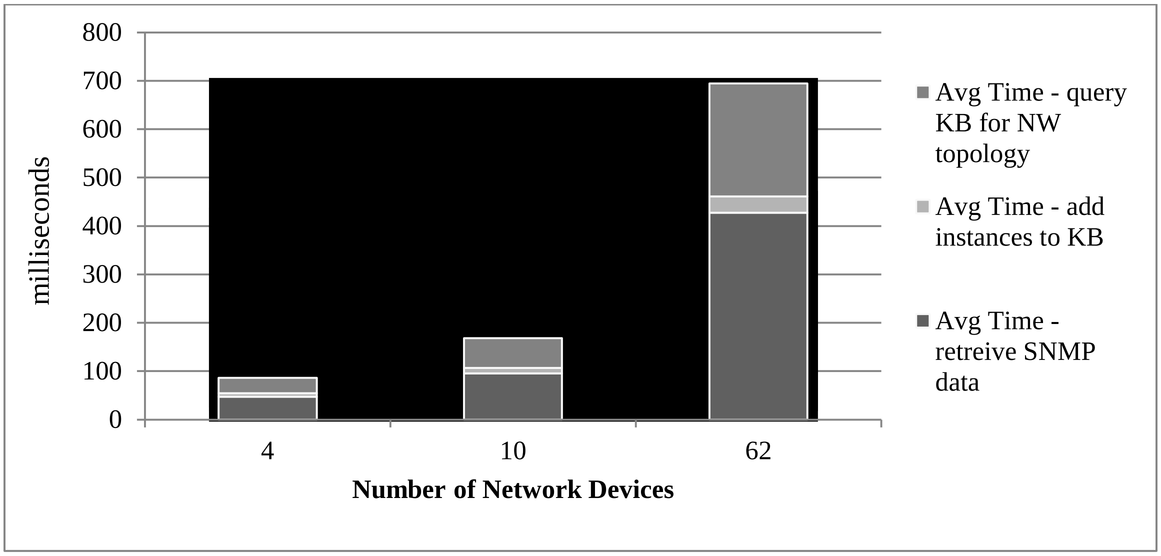


Figure 4.7: Results from the HMNMS for a live university network.

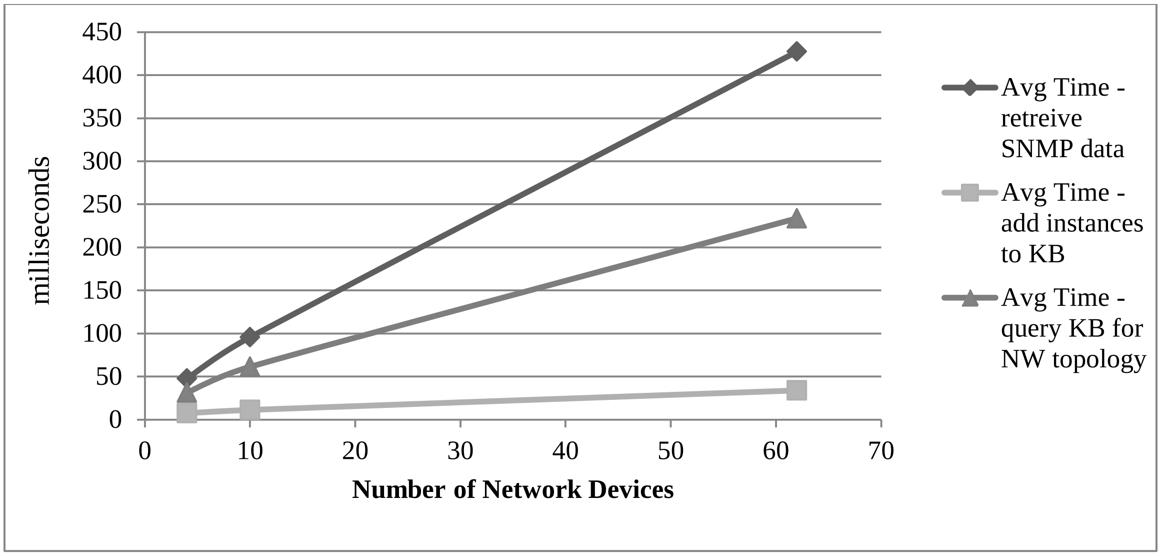


Figure 4.8: A scalability perspective of the sample query results from the HMNMS for a live university network.

4.5 Chapter Summary

Improved techniques represent a critical aspect of managing networks as they grow larger and more complex. As the network management task becomes more and more complex it becomes more difficult for humans to carry out this task. We already have networks of sufficient complexity that are subject to attack and cannot be properly managed in their entirety. As we have described by incorporating sufficient knowledge into an NMS and by unifying disparate networks through ontological representation we can begin to use computational power to address the network management problem.

In comparison to the alternative of manual processing of data, the overhead of obtaining the topology of a network with the new NMS is acceptable. The results of tests run to retrieve the network topology for a simulated network, a test network, and two live networks demonstrate that the overhead of the ontology (adding instances and retrieving query results) is minimal, particularly as the number of nodes is less than two hundred. If the network manager was responsible for an HMN and was not using this NMS, the manager would have to consult four different NMSs, one for each device type deployed. The manager would then have to manually combine all four network topologies returned in order to have one integrated network topology of the HMN. An obvious benefit of the NMS that uses ontology is the integration of diverse data.

Results of a network in a simulation environment and two live deployments show the HMNMS incurs negligible, acceptable overhead. The deployments in the live networks demonstrate the minimal set-up required to utilize the HMNMS. Thse two observations, adjoined with the benefits of the HMNMS, make it an obvious addition to the tool set of a manager of a HMN.

Chapter 5 An Analytical Model for Performance Analysis of a Heterogeneous Multi-tier Network

Network applications as a class of applications face many issues. These issues include response time, bandwidth capability, and connectivity. As a member of this class of applications an NMS has these same issues. The network manager must monitor and maintain the network but not impact the users' experience. To achieve this goal, it is vital to optimize the bandwidth, by minimizing the traffic overhead introduced by an NMS. In this chapter the performance of an HMN is analyzed while running an HMNMS.

The performance analysis provides a view of the impact of system design on network capacity. A key element of this analysis is determining if there are any bottlenecks in the HMN caused by the HMNMS. The analytical analysis was conducted for a heterogeneous, two-tier network, consisting of wired and ad hoc nodes.

5.1 Theoretical Analysis Based on Queuing Theory

The performance of the HMNMS was evaluated using the model proposed by Nishida [84]. Nishida developed an end-to-end performance model to conduct a bottleneck analysis. The end-to-end performance was defined as the accumulation of the processing time of all the components of the system. For this research, the end-to-end performance is defined as

$$T_{NMS} = T_{nd} + T_{ui} + T_{ont} + T_{int} + t \tag{5.1}$$

The components of the end-to-end performance are shown in Table 5.1 and correspond to the system components in the HMNMS, Fig. 5.1.

Table 5.1: End-to-End Performance Components [74].

Notation	Description	Corresponding System Component
T_{ont}	Processing time on the Ontology Sub-system	Ontology Sub-system
T_{int}	Processing time to add the ontology instances, representing the devices, to the knowledge base	Ontology Instances Interface
T_{ui}	Processing and Input/Output time of the UI	UI
T_{nd}	Processing time in the devices in the HMN	HMN devices
T	Transmission time to obtain the management data from the devices in the HMN	Links between Ontology Instances Interface and the HMN

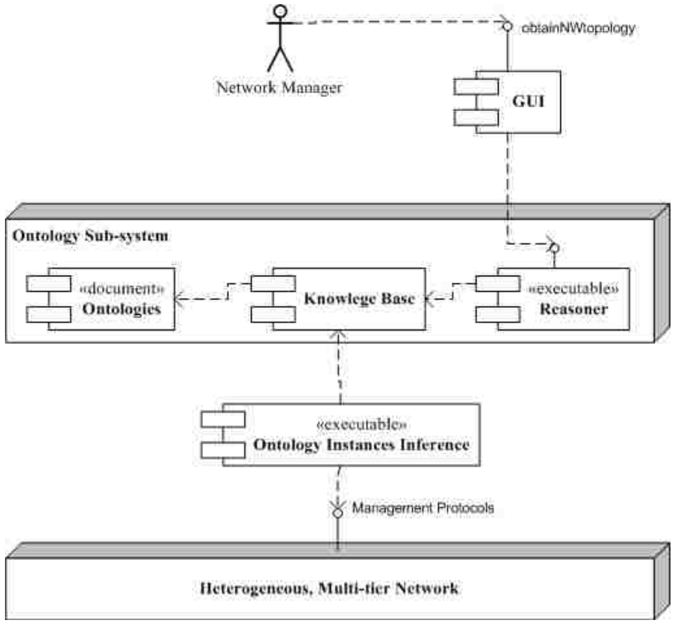


Figure 5.1: Component diagram of the Network Management System (NMS) [74].

The majority of the run time in the HMNMS is obtaining the data from the deployed devices and displaying it to the User Interface (UI). This is illustrated in Fig. 5.2. This portion of the run time consists of the node processing time, T_{nd}, the transmission time, t, and the UI processing time, T_{ui}. The wired and AHN portions of the network include sending an SNMP request to each node and receiving an a SNMP response. The round-trip time for the SNP request and response is the same for any NMS, including the HMNMS. The HMNMS uses existing protocols, such as SNMP, to retrieve the management data. Since the node processing time, T_{nd}, and the transmission time, t, are the same for any deployed NMS in a network, these two times are combined for the performance evaluation, T_d. The new formula is

$$T_{NMS} = T_d + T_{ui} + T_{ont} + T_{int} \tag{5.2}$$

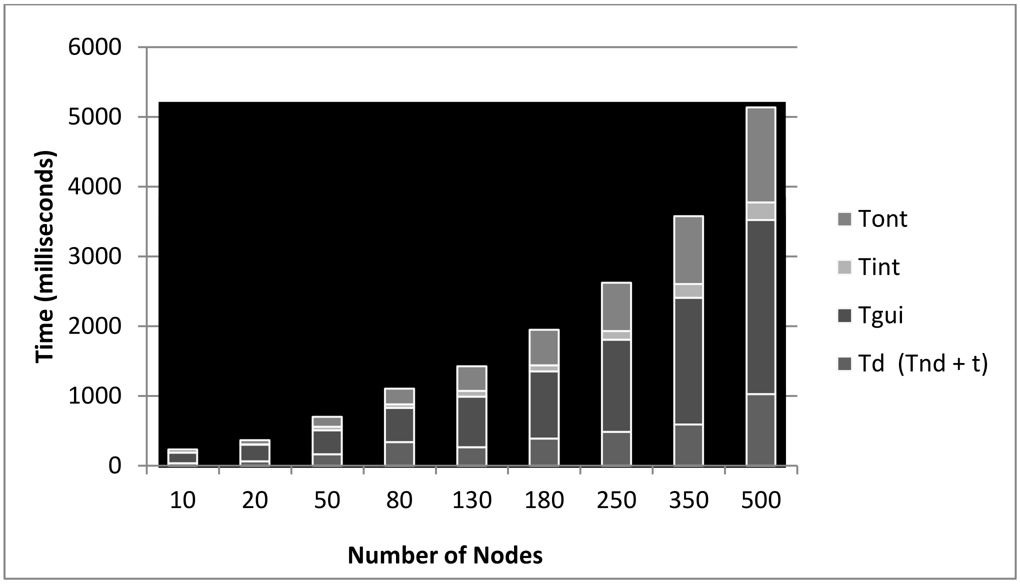


Figure 5.2: End-to-end performance times of experimental tests [74].

The other parts of this run time are the node processing time and the UI processing time. The UI processing time is required to display a graphical view of the network and the deployed devices to the network manager. The UI processing time is not unique to the HMNMS since any deployed NMS incurs the same overhead.

The overhead in the HMNMS that is new to this design is the Ontology Sub-system and the Ontology Instance Interface. For this reason, these two components, T_{ont} and T_{int}, are the key points for analysis. The goal is to minimize the processing overhead for these two components while maximizing the benefit of incorporating ontology into the NMS.

As observed in Fig. 5.3, the time to add instances of deployed nodes to the knowledge base, T_{int}, is reasonably small and almost constant. In the overall running time of the NMS, this time is negligible for two reasons. First, the time required to add the instances to the knowledge base for nodes in the network is insignificant. The second reason is due to the way the instances are added to the knowledge base. Instances are added to the knowledge base when deployed devices are identified in the network, which is a one-time occurrence during the running of the HMNMS. Currently, deployed devices are found by hard-coded addresses in the HMNMS. Future work will utilize some type of auto discovery of the devices. After the initial loading of devices to the knowledge base, all deployed devices exist in the knowledge base. After this initial loading, new instances are only added as new devices are deployed to the network. Since devices are added randomly, there typically is not a time when there is substantial overhead in the HMNMS due to new instances being added to the knowledge base.

The time to retrieve the network topology, T_{ont}, is exponential to the number of nodes, as shown in Fig. 5.3. As the number of nodes increases, the size of the knowledge base increases, so additional time is required to process and retrieve the instances. This property impacts the HMNMS one time, when the initial network topology is discovered.

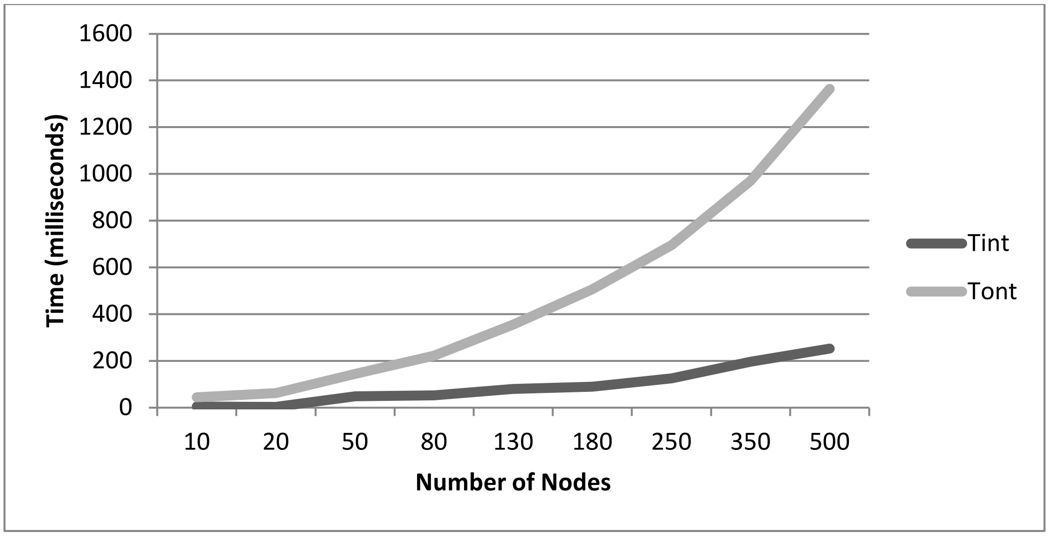


Figure 5.3: Ontology sub-system and instance interface times [74].

An analytical model was developed for the performance analysis of the HMNMS [85]. The HMNMS has two main systems: the User-Ontology System and the Management-Query System. The User-Ontology System is invoked when the network manager asks a query and has no impact on the Management-Query System. For instance, if the network manager wants to know the address of all deployed devices, the query is sent to the knowledge base and the response is returned to the network manager via a UI. The knowledge base query and response (T_{ont}) and the UI display (T_{ui}) are both components of the User-Ontology System. The User-Ontology System is separate from the Management-Query System and does not impact the end-to-end performance of the Management-Query System.

The Management-Query System queries the deployed devices and receives responses containing management information. The Management-Query System is the interaction between the knowledge base and the HMN. This interaction is the task of the Ontology Instances Interface.

The Management-Query System is evaluated using a queuing model for the end-to-end performance. The end-to-end performance for the Management-Query System is

$$T_{NMS} = T_d + T_{int} \tag{5.3}$$

The Ontology Instances Interface sends a query to all the nodes, which send responses back. The Ontology Instances Interface then adds a new instance or updates an existing instance in the knowledge base. Various implementation tests, which are discussed in the next section of this work, reveal that the overhead of the Ontology Instances Interface is negligible.

The HMN is modeled as a packet network. For the analysis, it is assumed that there is no network congestion. The queue at each device in the network is assumed to be an independent queue. It is assumed that all packets, both queries and responses, have the same size and priority when processed at each device. The Poisson distribution (explained briefly in section 2.3) is assumed for packet arrivals. The HMNMS is a request/response application with each device being managed generating one request and one response packet. This request/response query to each managed device is viewed as an independent packet flow. Each independent packet flow traverses a node twice, once for the request and once for the response. The average end-to-end delay for each packet flow (also referred to as flow here) is the sum of all delays of queues the flow traverses. Table 5.2 briefly explains many of the parameters used in the development of the analytical model.

Table 5.2: Analytical Model Parameters.

Notation	Description
P	Set of all flows in the network
p	An individual flow, $p \epsilon P$
x_p	Arrival rate of each flow
i	An individual node
$c_{p,i}$	Times a flow may traverse a node i
λ	Packet arrival rate
μ	Packet processing rate
N	Average number of packets in a queue
T	Delay
L	Packet size
$W(\lambda)$	Average packet delay caused by multi-access communication
I_p	Set of all nodes traversed by flow p
J_p	Set of all gateways traversed by flow p

Each flow may traverse a node i $c_{p,i}$ times, where $c_{p,i} \epsilon \{0, 1, 2\}$. The value of $c_{p,i}$ is:

- 0 if a flow never traverses node i
- 1 if the flow p traverses an end device i and returns
- 2 if the flow traverses a node both on its enquiring and responding paths

The set of all flows that traverse any given node i in the network is denoted as P_i, where $P_i \subseteq P$. The total packet arrival rate λ_i for a node i is written as:

$$\lambda_i = \sum_{p \in P_i} c_{p,i}\, x_p \tag{5.4}$$

The Kleinrock Independence Approximation [86] is an approximate analysis of networks of M/M/1 queues (M/M/1 queues are explained in section 2.3). The Kleinrock Independence Approximation asserts that all queues in the network can be modeled as a M/M/1 queue. The average delays in a network can be approximately calculated by assuming the delays in the queues are independent. The average number of packets in queue i can be expressed as:

$$N_i = \frac{\lambda_i}{\mu_i - \lambda_i} \tag{5.5}$$

Here μ_i is the packet processing rate of node i. If the propagation delay is ignored, then Little's Theorem is applied and the average packet delay is written as:

$$T_i = \frac{N_i}{\lambda_i} = \frac{\frac{\lambda_i}{\mu_i - \lambda_i}}{\lambda_i} = \frac{1}{\mu_i - \lambda_i} \tag{5.6}$$

A multi-access network is a network where multiple nodes access the same channel, such as an Ethernet or wireless channel. In such a network contention among nodes competing for the same channel will cause a delay. From the conclusion in [86], for a slotted CSMA/CD network, the approximated average packet delay caused by multi-access is expressed as:

$$W(\lambda) = \frac{\lambda\,\overline{X^2} + \beta(A+2\lambda)}{2[1-\lambda(1+B\beta)]} \tag{5.7}$$

Here λ is the total arrival rate to the bus from the nodes. The propagation and detection delay required for all sources to detect an idle channel after a transmission ends is

$$\beta = \tau\, C / L \tag{5.8}$$

Here β is expressed in terms of packet transmission units. τ is this time in seconds, C is the raw channel bit rate, and L is the expected number of bits in a data packet. $\overline{X^2}$ is the mean-square of the packet duration and is expressed as

$$\overline{X^2} = \sum x^2\, Pr.(X = x) \tag{5.9}$$

Recalling the assumption that all management request and response packets have identical lengths, $\overline{X^2}$ is simply X^2. The values of A and B, two constants, depend on the detailed assumptions of the network (see [87]). This delay W(λ) can be added to the delay of any flow going through a multi-access gateway.

The total delay of flow p can be expressed as:

$$T_{p,total} = \sum_{i \in I_p} T_i + \sum_{j \in J_p} W_j(\lambda) \tag{5.10}$$

The actual average end-to-end delay depends on the topology of the network. The topology of a general HMN (Fig. 5.4) can be generalized as in Fig. 5.5.

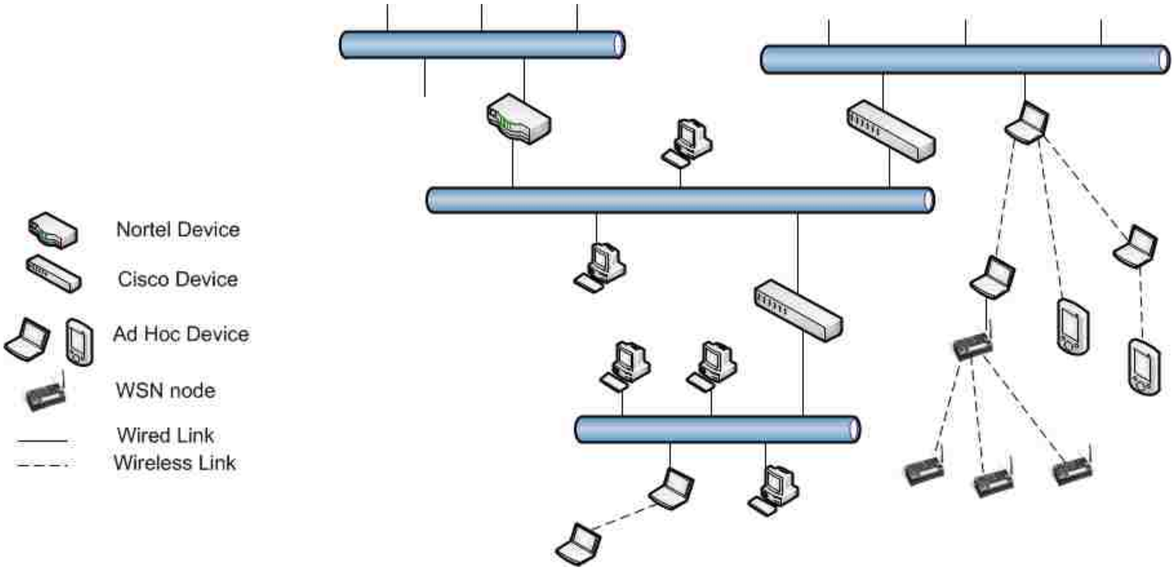


Figure 5.4: A general Heterogeneous Multi-tier Network.

The queuing delay caused by the switch in Fig. 5.5, which is part of the wired tier, is:

$$T_{wired} = \frac{1}{\mu_{wired} - \lambda_{wired}} = \frac{1}{\mu_{wired} - x_{wired}} \tag{5.11}$$

where x_{wired} is the data rate of the switch query flow. The queuing delay caused by the

Ethernet gateway is:

$$T_{eth-gw} = \frac{1}{\mu_{eth-gw} - \sum_p 2x_p} \qquad (5.12)$$

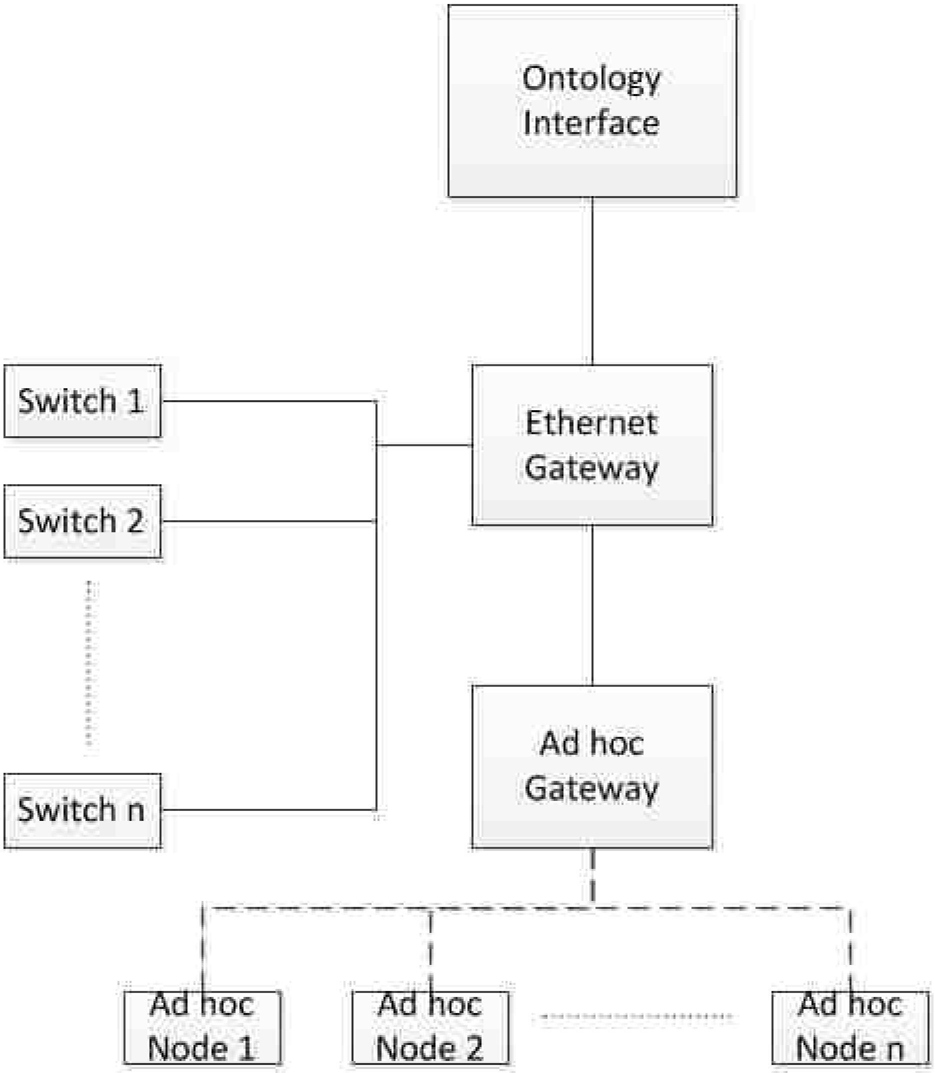


Figure 5.5: A generalized network topology.

Since the system is a request/response system, all flows traverse the Ethernet gateway, the Ontology Instances Interface, the Ontology Subsystem and the User Interface twice. The equations for the delays calculate delays in the ideal case, which assumes the management packets are always given top priority by the operating system. The actual delay will vary slightly depending on how the operating system schedules packets to be forwarded.

The total average end-to-end delay for inquiring a wired node is:

$$
\begin{aligned}
T_{wired\ total} &= T_{wired} + T'_{eth-gw} + T_{int} + W_{eth-gw}\left(\lambda_{eth-gw}\right) \\
&= \frac{1}{\mu_{wired} - x_{wired}} + \frac{1}{\mu'_{eth-gw} - \sum_p 2x_p} \\
&\quad + \frac{1}{\mu_{int} - \sum_p 2x_p} + W_{eth-gw}\left(\sum_P x_p\right) \qquad (5.13)
\end{aligned}
$$

T_{wired}, T'_{eth-gw}, T_{int} are delays caused by wired devices, the Ethernet gateway, and the Ontology Interface, respectively. T'_{eth-gw} is the delay caused by the Ethernet gateway to forward packets. This delay is different from T_{eth-gw} because forwarded packets will send interrupts to the processor, causing additional overhead to these packets. This is due to the fact that the operating system will interrupt their processing, causing them to be in placed in the processor queue, incurring some queuing delay. An ad hoc gateway behaves in the same manner.

Assume there are m*wired* wired devices and m*adhoc* ad hoc nodes in the network. The total average end-to-end delay for sending and receiving management packets to an ad hoc node is:

$$T_{adhoc\,total} = T_{adhoc} + T'_{adhoc-gw} + T'_{eth-gw} + T_{int}$$
$$+ W_{adhoc-gw}(\lambda_{adhoc-gw}) + W_{eth-gw}(\lambda_{eth-gw})$$
$$= \frac{1}{\mu_{adhoc} - x_{adhoc}} + \frac{1}{\mu'_{adhoc-gw} - 2m_{adhoc}x_{adhoc}}$$
$$+ \frac{1}{\mu'_{eth-gw} - \sum_p 2x_p} + \frac{1}{\mu_{int} - \sum_p 2x_p}$$
$$+ W_{adhoc-gw}(\sum_P m_{adhoc}x_{adhoc})$$
$$+ W_{eth-gw}(\sum_P x_p) \qquad (5.14)$$

For the example network topology illustrated in Fig. 5.5, $\sum_P 2x_p = 2m_{adhoc}x_{adhoc} + 2m_{wired}x_{wired}$. The summation is over 2x$_p$ because all flows traverse the Ethernet gateway, Ontology Inferences Interface, Ontology Subsystem, and User Interface twice.

5.2 Performance Analysis of a Heterogeneous Multi-tier Network

A performance analysis of the capacity of the Management-Query System was performed. The number of wired or ad hoc nodes that can be supported while providing a reasonable query response time was determined.

From the end-to-end delay expressions for wired devices (Eq. 5.13) and ad hoc nodes (Eq. 5.14), the total delay for a query flow can be calculated. The total delay is the sum of the delays of each device along the path. As a result, the system capacity is reached when any device in the path reaches its capacity. These devices are the two gateways in Fig. 5.5. In this case, any query flow traversing one of these devices can have infinite delay, possibly causing packet loss.

To maintain a stable system, each term on the right hand side of Eq. 5.14 cannot go to infinity. To prevent this, the denominator of each term cannot be greater than 0. Because the Ethernet gateway is traversed by all traffic in the network, it is most likely to be the bottleneck. To keep $T_{eth\text{-}gw}$ finite, the number of wired and ad hoc nodes that can be supported must satisfy:

$$\mu_{eth-gw} - (m_{adhoc} + m_{wired})2x_p > 0 \tag{5.15}$$

This can be written as:

$$m_{adhoc} + m_{wired} < \frac{\mu_{eth-gw}}{2x_p} \tag{5.16}$$

Similarly, to keep the ad hoc gateway delay bounded, the number of ad hoc nodes should satisfy:

$$\mu_{adhoc-gw} - m_{adhoc}2x_p > 0 \tag{5.17}$$

This can be transformed into

$$m_{adhoc} < \frac{\mu_{adhoc-gw}}{2x_p} \tag{5.18}$$

The maximum number of ad hoc and wired nodes that can be supported by the Ethernet gateway is expressed by Eq. 5.16. The maximum number of ad hoc nodes that can be supported by the ad hoc gateway is expressed by Eq. 5.18. The capacity of the network is determined by both equations, requiring both equations to be satisfied at the same time. This is a theoretical prediction of the capacity. In a practical situation, the constraint may vary due to the dynamic nature of the hosts and network, such as the operating system scheduling policy and traffic patterns.

5.2.1 Implementation of the Model

A test network was deployed to verify the theoretical analysis with empirical results. The focus of this analysis was the ad hoc tier of the network since the ad hoc gateway was determined to be a critical node.

The test ad hoc network consisted of three Linux laptops with 802.11g wireless network cards. One of the laptops was the ad hoc gateway and the other two were ad hoc nodes connected to the ad hoc gateway wirelessly. A desktop was the management station. The management station was connected directly to the ad hoc gateway with a network cable to eliminate the Ethernet gateway performance fluctuations. The management station periodically sent SNMP query packets to the three ad hoc nodes, including the ad hoc gateway. This simulated the operation of a management station in a live network.

To simplify the analysis, the processing rate for all the deployed devices is assumed to be identical and the delays caused by multi-access communication are ignored. The value used for the processing rate was determined from experimentation and was 1/20 packet per millisecond. The query rate in the test network is constant. This differs from the theoretical analysis, which follows a Poisson distribution.

The delay caused by the ad hoc gateway and one of the ad hoc nodes is shown in Fig. 5.6. The delay caused by the ad hoc gateway increases faster than the delay caused by the ad hoc node. The primary reason for this is because all packets to the ad hoc tier flow through the ad hoc gateway. Also, as the gateway is processing packets, the operating system assigns different priorities to the packets destined for the gateway and packets to be forwarded by the gateway. This is explained later in this section.

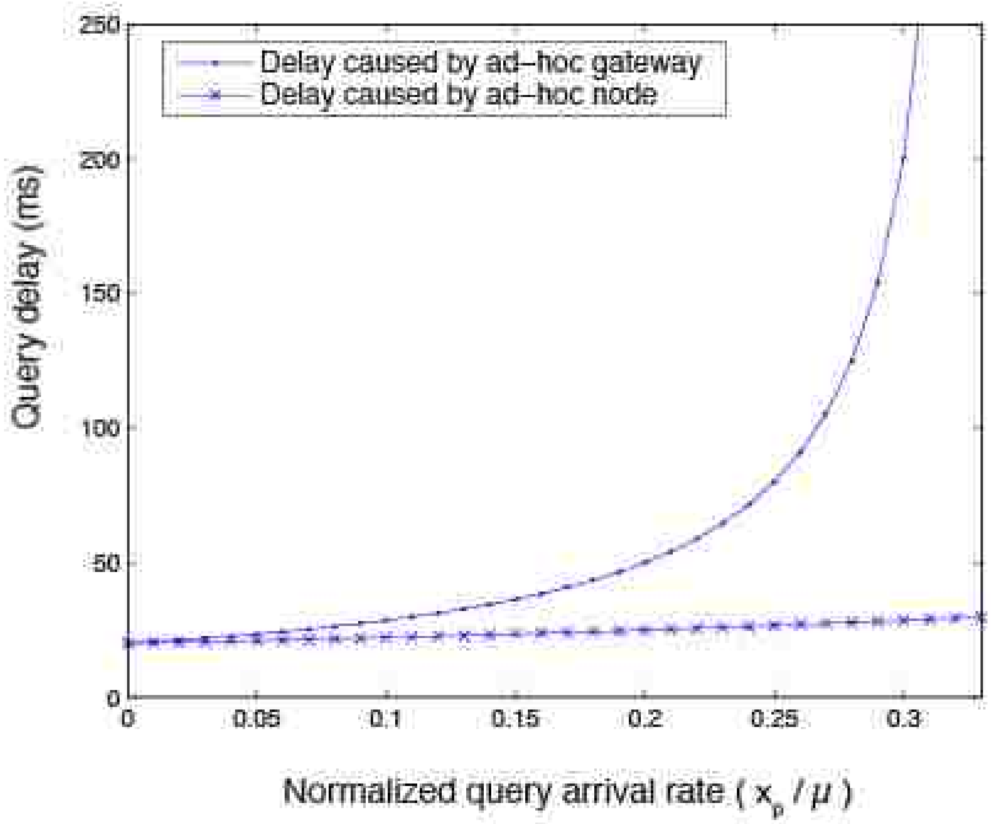


Figure 5.6: Delays caused by the ad hoc gateway and nodes from the theoretical analysis [85].

Additional test results for the ad hoc network are shown in Fig. 5.7 to 5.9. The x-axis in these figures is the indices of the query packet for each device in the network. The indices are ordered in time sequence. The y-axis is the delay for the management query and response. The figures show the query delay for each of the ad hoc nodes for varying inter-arrival times of the query packet. Fig. 5.7 to 5.9 show the queuing delay for the ad hoc gateway, one ad hoc node, and the second ad hoc node, respectively.

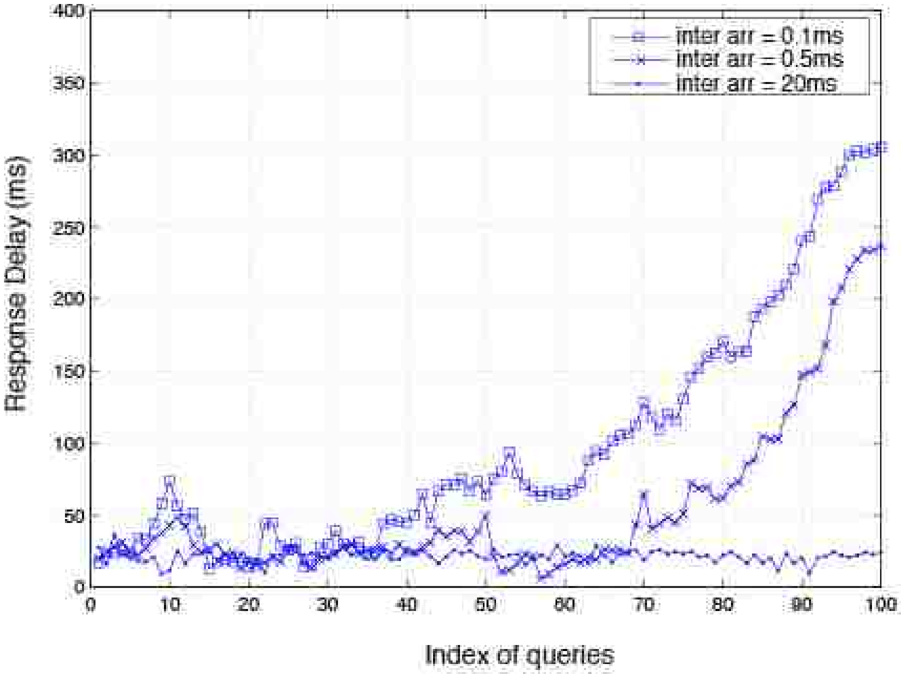


Figure 5.7: Query delays at the ad hoc gateway [85].

Several observations can be made from the experimental results. The first observation is the buffering effect. When the tests began there were fewer packets in the network so the delay was smaller. As more packets are injected into the network, packets are buffered and the queuing delay increases.

The second observation is that the queries for the ad hoc gateway have a higher average delay than the packets for the ad hoc nodes. This is because of how the node handles incoming packets. When the node's network card receives an Ethernet frame, an interrupt is sent to the operating system. The operating system stops the current process on the processor to handle the interrupt. If the packet is to be forwarded, then the operating system will immediately forward the packet. This causes any packet destined for the ad hoc gateway and currently being processed to be preempted and placed in a processor queue causing the packet being processed to incur some additional processor queuing delay. Utilizing an ad hoc gateway that has multiple processors may decrease or eliminate this additional processor queuing delay. If one processor is currently processing a management packet, another processor may be able to handle the interrupt and process the packet to be forwarded.

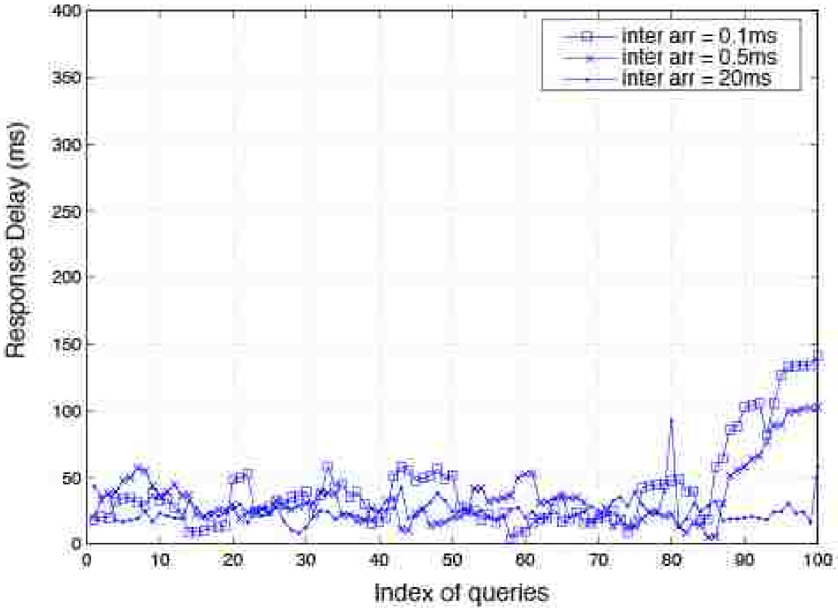


Figure 5.8: Query delays at the ad hoc node 1 [85].

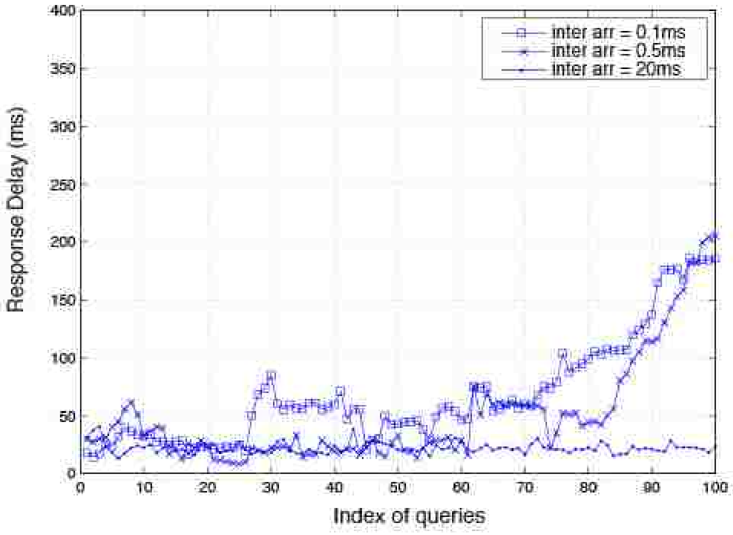


Figure 5.9: Query delays at the ad hoc node 2 [85].

In a traditional UNIX operating system, the scheduler categorizes tasks into five different categories. Each category has a different priority. These five categories, in decreasing priority order, are [88]:

- Swapper
- Block I/O device control
- File manipulation
- Character I/O device control
- User processes

This scheduling scheme is intended to provide the highest priority for I/O operations. Forwarding packets is an I/O task with higher priority over local MIB checking tasks, which are categorized as user processes. When the operating system receives an interrupt, it will stop the processor and suspend the process currently using the processor. This behavior adds to the larger average round trip delays for the SNMP packets destined for the ad hoc gateway.

The conclusion from the experimental results is that the ad hoc gateway is the bottleneck of the ad hoc tier. This is supported by the figures, which show the higher growth rate of the delay for the ad hoc gateway compared to the ad hoc nodes. As the amount of network traffic increases, the delay for the ad hoc gateway is significantly higher than the ad hoc nodes. The delay for packets querying the ad hoc gateway increases faster than other packets and eventually packet loss will occur. This experimental result confirms the conclusion from the theoretical analysis.

5.3 Chapter Summary

In comparison to the alternative of manual processing of data, the overhead of obtaining the network topology is acceptable under conventional use. If the network manager is responsible for an HMN and is not using the HMNMS, the manager must consult four different NMSs, one for each device type deployed. To obtain one integrated network topology of the HMN, the manager must manually combine all four network topologies returned by the various NMSs. An obvious benefit of the HMNMS that uses ontology is the integration of diverse data. The benefits would be evident to any network manager that must manage a HMN. The results of the experiments conducted show that the overhead of incorporating ontology into the NMS are acceptable given the benefits provided for the topology discovery of a HMN, provided no path devices reach their capacity.

A theoretical analysis was performed to provide an analytical view of the network performance. The theoretical analysis provides insight to deployment considerations for the HMNMS. Specifically, two deployment parameters are considered in the analysis, the inter-query time and the number of nodes that can be supported by one ad hoc gateway.

The inter-query time is the amount of time between the management station sending queries to a deployed node. The inter-query time must be small enough to obtain accurate information from the nodes for proper management but not too small that the queries inject too much traffic into the network. The analysis concludes that the gateways in the network are the bottlenecks of the query flow. A test AHN was deployed to conduct experiments for query delays. The experimental results support the theoretical conclusion showing that the ad hoc gateway is the bottleneck.

Chapter 6 **A Formal Representation for Complex Attacks using Ontology**

A characteristic of Intrusion Detection Systems (IDSs) is that they are optimized to identify simple attacks. A simple attack is an attack against a host or networking consisting of a single step. Examples of a simple attack are a ping scan, where an attacker scans IP addresses in a network to find active hosts, or a denial of service attack, where an attacker takes a host or network offline by making it unavailable to users.

Often an attack against a network consists of several stages, with each stage being a simple attack. An attack consisting of multiple stages of simple attacks is a complex attack. A complex attack can be defined as a combination of two or more simple attacks or two or more complex attacks in a spatial or temporal domain.

Complex attacks often require the examination of both their temporal and spatial domains for identification. The temporal domain for an attack requires the examination of the time period when the attack occurs. During this time period, there may be multiple events that indicate a complex attack has occurred.

The spatial domain is the location, either physical or logical, in the network where the attack occurred. Multiple events in the same network or subnet may indicate a complex attack, while the same events in different networks or subnets may indicate normal traffic. For example, consider a user performing troubleshooting on their host because they are experiencing connectivity issues. The user may try to ping several different hosts throughout their network and the Internet to determine the source of the connectivity problems. This is legitimate network traffic; however, if the user pings many hosts on the same network, this may indicate the user is attempting to find active hosts, the first step in many complex attacks. For an IDS to properly identify a complex attack, it is necessary for the system to identify an attack based on multiple events that occurred in multiple locations on the network over a period of time.

Another aspect of the spatial domain is from the source address. The simple attacks comprising one complex attack may originate from different hosts, thus different source addresses. An attacker may simply be using various hosts to initiate each simple attack, or it may be several attackers collaborating on the complex attack.

Many times, an attacker conducts some preliminary actions before initiating a complex attack. Consider the following example. An attacker uses a port scanner tool, like nmap [89], to find open telnet or ssh ports on hosts. The attacker will then telnet/ssh to these hosts and view the banner or motd. If the banner/motd contains the string “User Access Verification”, this indicates a Cisco router. The attacker then uses a tool like SING [90] to create a custom ICMP (Internet Control Message Protocol) packet for a netmask request (ICMP type 17). Typically only routers respond to an ICMP type 17 request. The attacker will then attempt to connect to SNMP on the router by using common SNMP community strings. The attacker may then take advantage of known vulnerabilities for the device, download the entire configuration for the device, and possibly even modify the device configuration. When the router is attacked, it may lead to valuable information to allow the attacker to attack more critical information/servers, or allow the attacker to disable the entire network.

As another example, illustrating the need to consider all traffic and attack attempts and not just attacks aimed at vulnerabilities of specified nodes, consider a vulnerability on port 80 of a webserver. If the server is not a web server or if the firewall has port 80 blocked to that server, then it may not be a critical vulnerability to the network manager. But what if the firewall was previously compromised and the firewall rules were changed or removed by the attacker? What if a user installs a new web server on a host that is available through the firewall? Now there will be traffic on the network to port 80 of that server, with an external source IP address, which may indicate to the network manager that the firewall is compromised. The network manager must examine data on the network for all types of attacks, including successful attacks and attack attempts.

Consider an example of the need to examine the temporal domain of attacks. One of the early steps of many complex attacks is for the attack to identify ports that are open on devices. If the network manager observes traffic to determine if one port is open on a server, this indicates very little about the possible occurrence of a complex attack; however, observing a check for multiple ports in sequence, may indicate a complex attack is occurring or has occurred.

A Reasoning Intrusion Detection System (RIDS) utilizes reasoning (primarily inference) in attack identification. The reasoning mechanisms and associated knowledge base are used to provide efficient and reliable analysis of collected network data to aid in attack identification. The reasoning capability of a RIDS also provides the ability to identify a family of generic attacks. The approach we are taking to augment the typical IDS is to add an ontological representation of the network space along with a reasoning engine to operate on the ontology [91]. The result is a Reasoning Intrusion Detection System using Ontology.

All ontologically-based systems have the ability to make inferences using the knowledge contained in the ontology. In fact this is where their tremendous power lies. Through the use of ontological knowledge we are able to carry out complex analysis on data collected from the network. In addition the ontology can grow and change as time progresses because of the rapid change in networking and networks. For example if we understand network traffic from a certain deployed virus than we can use that information to augment the ontology in such a way as to recognize that and similar viruses.

Therefore one tremendous advantage gained by the ontology is the inference capability provided allowing additional knowledge to be learned. This will allow the incorporation of new rules into the identification process, allowing the IDS to use the meaning of the network data to help identify attacks. For example, if a port scan follows a ping scan, within a specified amount of time, it may indicate the occurrence of a complex attack. As another example, consider what happens when an attacker conducts a denial of service attack on a host using the ping utility. This attack results in the creation of a *PingFlood* instance in the knowledge base. In the ontology (see section 6.3), the *PingFlood* class is a subclass of *Flood*, which is a subclass of *Resources*, which is a subclass of *DoS* (so indirectly, *PingFlood* is a subclass of *DoS*). By using ontology, a query for all DoS attacks returns the newly created instance for the ping flood attack. Without the inference provided by ontology this query would only return direct instances of the *DoS* class, which would not include the ping flood instance.

The inference provided by ontology allows more advanced information to be learned from the network data. For example, if there are multiple port scans found in the data collected, and they occur within a specified time frame, then the ontology can infer that a port scan occurred. The reasoning will then identify the various complex attacks that have a port scan as one of its attack elements. Without the use of reasoning, this would require a sophisticated, difficult-to-maintain program or manual analysis.

Another important advantage is the semantic expressiveness provided by ontology. XML and XML Schema provide structure to information but no semantic information. RDF Schema provides limited semantics, but not sufficient semantics for a RIDS. For example, RDF Schema does not provide for disjoint classes, i.e., a packet cannot be both TCP and UDP. RDF Schema also does not provide the ability to specify cardinality restrictions. For example, an instance can have only one source address; this limitation can be specified in ontology using cardinality statements. A powerful semantic expressiveness exploited by this research and not supported by RDF Schema is the Boolean combinations of classes. The formal representation developed in this research creates new classes by combining other classes using Boolean operators, such as union and intersection. It is important to note that there is a trade-off between high expressivity and computation costs. This will be discussed in more detail in chapter 7.

6.1 Generalized Attack Trees

A complex attack consists of multiple events or attack elements. Decomposing a complex attack into its individual attack elements provides a better understanding of how attackers launch complex attacks. This analysis provides the ability to consider other related complex attacks that may consist of similar elements and produce similar results. Knowledge of specific attacks can lead to the discovery of a more comprehensive set of generic attack descriptions.

Individual attacks were examined to determine if aggregate sequences are represented in the wild. For example, a Man-In-The-Middle (MITM) attack is a group of unrelated, individual attacks that act together. To develop a generic attack tree, a specific MITM attack was launched and the data studied. The specific MITM attack consisted of the following individual attacks:

1. A ping scan against the network
2. A SYN scan to find open TCP connections on an active host on the network
3. A series of TCP connections against an active host on the network to predict the TCP sequence number
4. A Denial of Service (DoS) attack against the second host in an established TCP connection by sending many pings to the host
5. Spoof the IP address of the second host in the TCP connection

By examining these individual attacks, and looking at other MITM attack data, it was determined that each step can be generalized. For example, the second step, a node scan to find active TCP connections, can be done in a number of ways. This example used an SYN scan, but another MITM attack may use an FIN scan. Also, for the fourth step, there are many different ways to launch a DoS attack against a host, including the ping packets used in this example, SYN packet flood, application floods, etc. The generalized steps were aggregated into a generic attack tree for the MITM attack, which can be used to identify many different types of a MITM attack, even as new ones develop.

Attack trees are used to aid in complex attack identification. An attack tree is a tree diagram representing the steps in a complex attack. A single path in the attack tree illustrates the steps for a particular complex attack. With multiple paths in each attack tree, multiple specific complex attacks are represented. The root node of the tree represents the goal of the attack. The other nodes represent the steps necessary to reach the goal. The nodes are joined by the "AND" keyword to indicate that each node is required to reach the goal. Same-level nodes not linked by the "AND" keyword represent options for that particular step. Most attack trees, such as the ones used in this work, have "AND" conditions for all the root's children, indicating that each of root's children must be satisfied for the goal to be achieved. Each branch from each child node from root then indicates a method to satisfy that child node.

For example, to take a host in a TCP connection offline for the duration of the connection, an attacker may execute a denial of service attack against the host or spoof the host's MAC (ARP) address. This branch of an attack tree is illustrated in Fig. 6.1, showing two different methods to spoof a host's MAC address. Each of these options would be represented in the attack tree as child nodes of the same parent (the parent node would be "take host offline for a TCP connection") with no "AND" connection, indicating success of one of the child nodes would satisfy the parent node.

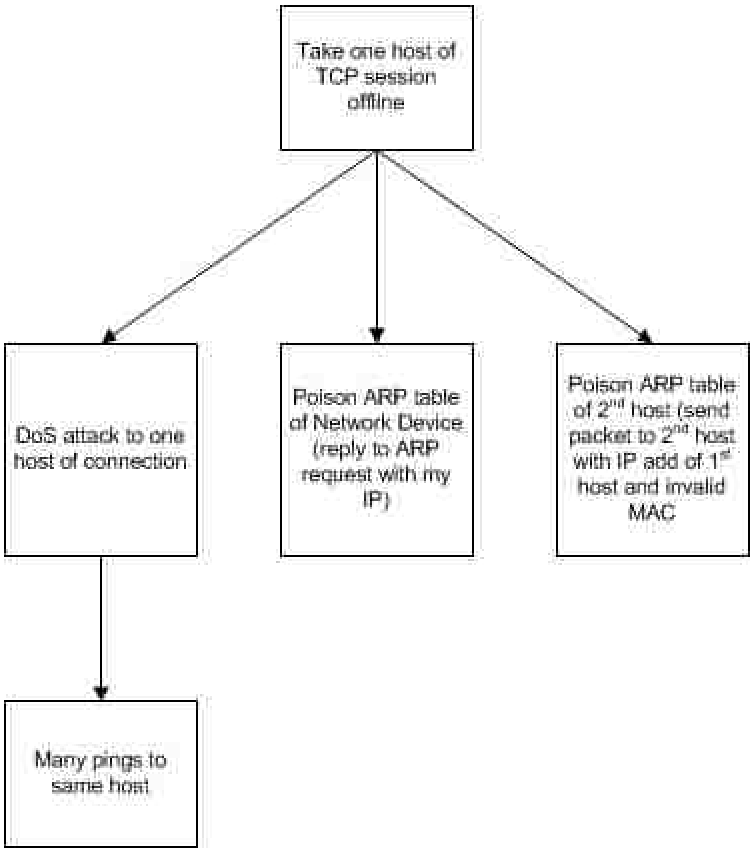

Take one host of TCP session offline
DoS attack to one host of connection
Poison ARP table of Network Device (reply to ARP request with my IP)
Poison ARP table of 2nd host (send packet to 2nd host with IP add of 1st host and invalid MAC
Many pings to same host

Figure 6.1: An attack tree branch example.

The attack trees used in this research began as attack trees for specific attacks. Common complex attacks were identified as a sequence of simpler attacks and the attack trees were constructed from these simpler attacks. The specific attack trees were studied; similarities were identified, which lead to the development of generalized attack trees. A generalized attack tree is a representation of a class of complex attacks. These generalized attack trees were used to develop the formal representation and were based on the Department of Defense's (DoD's) five pillars of Information Assurance [92]. The five pillars are confidentiality, integrity, authentication, non-repudiation, and availability. As the generalized attack trees were developed, it was discovered that each root node matched one of the five pillars. The result was four generalized attack trees, each corresponding to one of the five pillars. The methods used by attacks to breach confidentiality and non-repudiation are similar resulting in one generic attack tree for these two pillars.

An example specific attack tree is depicted in Fig. 6.2. Each node was manually assigned a unique identification number, which is used by the RIDS. Many of the attack elements in a variety of attack trees are similar. Many attacks include an attacker first finding all available hosts on a network (a ping scan) and then finding all the open ports (a port scan) on each available host. Similar attack elements were identified as generic simple attacks. If a node in an attack tree is one of the generic simple attacks, then the node is mapped to that attack.

A mapping was manually developed for the generic attack trees. For example, the first step in Fig. 6.2 is the “find active hosts on network”. This is a very common step in complex attacks and is found in all attack trees used in this research. This step is identified as generic attack #1. The corresponding node in Fig. 6.2, Node 8.1, is mapped to generic attack #1. Any time generic attack #1 is identified, it will color all corresponding nodes in attack trees, such as Node 8.1 in Fig. 6.2.

When identifying attacks, the IDS identifies all the generic attacks and then identifies each node in the attack trees that correspond to these generic attacks, based on mappings developed. These nodes are marked in the attack tree based on the coloring scheme described in the next section. The IDS then identifies any specific attacks, which do not map to a generic attack, and annotates those nodes in the attack trees. The annotated attack trees are then used by the RIDS to assist with complex attack identification.

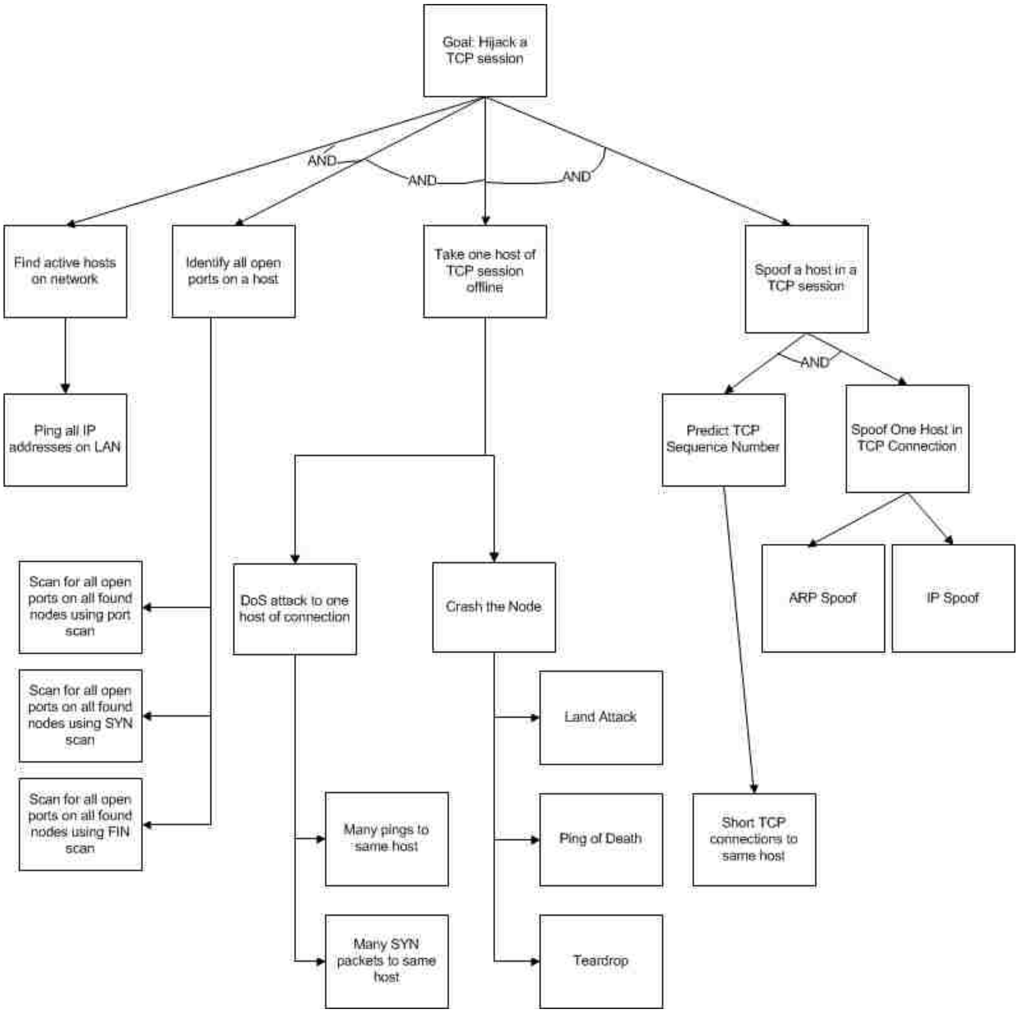

Goal: Hijack a TCP session
AND
AND
AND
Find active hosts on network
Identify all open ports on a host
Take one host of TCP session offline
Spoof a host in a TCP session
AND
Ping all IP addresses on LAN
Predict TCP Sequence Number
Spoof One Host in TCP Connection
ARP Spoof
IP Spoof
Scan for all open ports on all found nodes using port scan
Scan for all open ports on all found nodes using SYN scan
Scan for all open ports on all found nodes using FIN scan
DoS attack to one host of connection
Crash the Node
Land Attack
Ping of Death
Teardrop
Many pings to same host
Many SYN packets to same host
Short TCP connections to same host

Figure 6.2: An attack tree example.

6.1.1 Plan Recognition and Attack Trees

Plan recognition, an Artificial Intelligence research area, is “the process of deducing an agent’s goals from observed actions” [93]. A hierarchical task network (HTN) is very similar to an attack tree. The use of attack trees for the ontology development is similar to a plan recognition problem. Future work may entail the development of HTNs in place of the attack trees and the evaluation of utilizing the HTNs to develop the ontology.

Geib [94] describes the complexity when a plan recognition system must consider multiple instances of the same goal. This is the case when describing complex attacks as there are many different methods an attack may utilize to launch a complex attack. Geib used the cyber security domain in his discussion of the complexity of a plan library consisting of multiple observations for the goal where the goal is a complex attack.

According to Kichkaylo, et.al. [95], the assumptions of traditional plan recognition to the intrusion detection domain are not valid. Geib [94, 96] believes that it is valid but more complex. Beyond the complexity reason described above, another reason for this complexity is the fact that attackers attempt to hide their actions. Many attackers will attempt to remove all evidence of their attack by removing entries in log files pertaining to their attack steps. Kichkaylo, et.al. and Geib both developed approaches based on plan recognition to help in detecting intrusions.

Detecting attacks after they occur is an important step in the security cycle; however, it would be optimal to predict an attack before it occurs. This is a very difficult endeavor as one cannot easily predict what an attacker may do in the future. Plan recognition may help with determining the path an attacker may take based on the current knowledge. Geib [96] outlines two problems that add to the complexity of using plan recognition in the intrusion detection domain. The first one is because attacks typically have multiple goals. The second problem is that many of the steps in a complex attack may also be a legitimate use of the network. For instance, many pings to nodes on one network is used by many attackers to find hosts that are active and open to an attack, but this may also be used by a network manager to help in diagnosing a network problem. Using probabilities can help alleviate these two problems.

Geib introduced a plan recognition algorithm to help predict an attack by using probability. This algorithm assigns probabilities for each goal to determine the top-level goals of the attacker. The algorithm is based on Combinatory Categorial Grammars (CGGs), a grammar formalism used in Natural Language Parsing (NLP). It may be appropriate to consider applying these techniques to TRIDSO but first their applicability must be explored. We defer this to future research.

6.2 Design of the Formal Representation

Many of the RIDSs detect attacks against hosts. Many attackers will target network devices, such as routers. If an attacker can breach a router, they can often gain valuable information about other nodes on the network or impact the entire network (cause the entire network to not function properly). The developed RIDS detects attacks against any node on the network.

Another characteristic of many of the RIDSs is that they detect attacks based on vulnerabilities. The RIDS will identify vulnerabilities against systems and evaluate the threats against these systems, based on the vulnerabilities and the current state of the system. For example, Undercoffer, et al [61, 62] developed a target-centric approach using ontology. In their work, the focus was on the target nodes and the state of the target nodes. This included the components of the nodes, such as the operating system, network layers, and processes running on the node. This requires monitoring of the nodes and their components. Mandujano, et al [63, 64] also monitored resources using agents installed on the nodes. These agents collected data on the nodes, such as program profiles. The work by Martimiano and Moreira [54, 55] assumed that all security incidents exploit a vulnerability. Their work assumed that an attacker used a tool that manipulated a known vulnerability.

The RIDS developed in this research does not focus on target nodes or vulnerabilities but will identify attacks based on network events. By examining network traffic, the RIDS can detect attacks regardless of existing vulnerabilities; it examines the traffic on the network and identifies events that may indicate attacks. It does not matter if there is a known vulnerability; if the traffic looks like a possible attack, it will be detected.

Another advantage of using traffic to identify attacks is the ability to also identify attack attempts. An attempted attack may be just as important to a network manager as a successful attack. Consider the scenario where an IDS simply watches for attacks against the web port of web servers. If a user installs a new web server on the network, this server would be vulnerable to an attack since the IDS is not aware of this web server. It may also be the case that this new web server is susceptible to vulnerabilities because it is not patched correctly. By analyzing network traffic, the RIDS in this research detects an attack attempt against the web port on any device on the network, and can alert the network manager. This also allows the network manager to see what types of attacks are being attempted against their network so they can properly secure the network and its resources.

Attacks and attack attempts are detected regardless of the state of the nodes on the network. Detections are performed based on observed traffic conditions and not the state of the nodes. Consequently, this RIDS does not require additional software to obtain information about the nodes. There is no extra installation or overhead on the nodes to utilize this RIDS.

Another advantage of using traffic to identify attacks is the ability to identify attacks against multiple hosts on the same network. Consider an attack attempting to find active hosts to attack. The first step for the attacker is to ping all hosts on the network by incrementally going through all IP addresses on the network. By examining all network traffic and not just traffic at specific hosts, all of the ping packets are observed. This results in an ping scan attack being detected.

6.3 Development of the Formal Representation Using Ontology

6.3.1 Traffic Representation

The traffic ontology, see Fig. 6.3, represents the raw network traffic data in a variety of forms. All network traffic is first added to the knowledge base by creating instances for all packets captured. The instances are created based on the data found in the packet. For instance, if the packet represents a TCP packet, then a *TCPPacket* instance is created.

The OWL code for the *TCPPacket* class is provided in Fig. 6.4. This only contains the properties specific to the *TCPPacket*; it will also inherit the properties from the *L4Packet*, *IPPacket*, *L2Packet* and *Packet* classes (the OWL code[1] for all the classes is provided in Appendix B).

[1] Available for download at http://faculty.kutztown.edu/frye/res/index.html

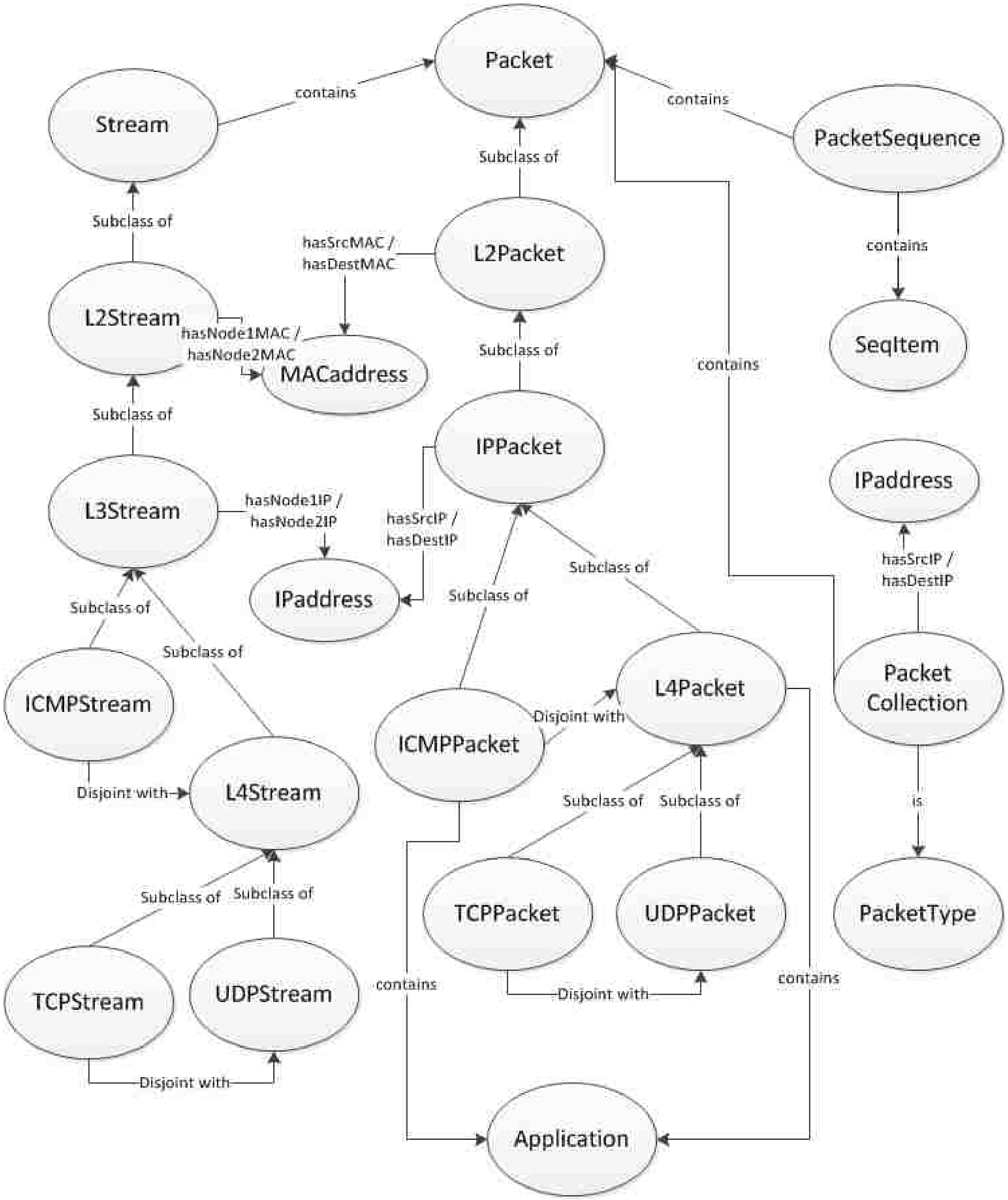
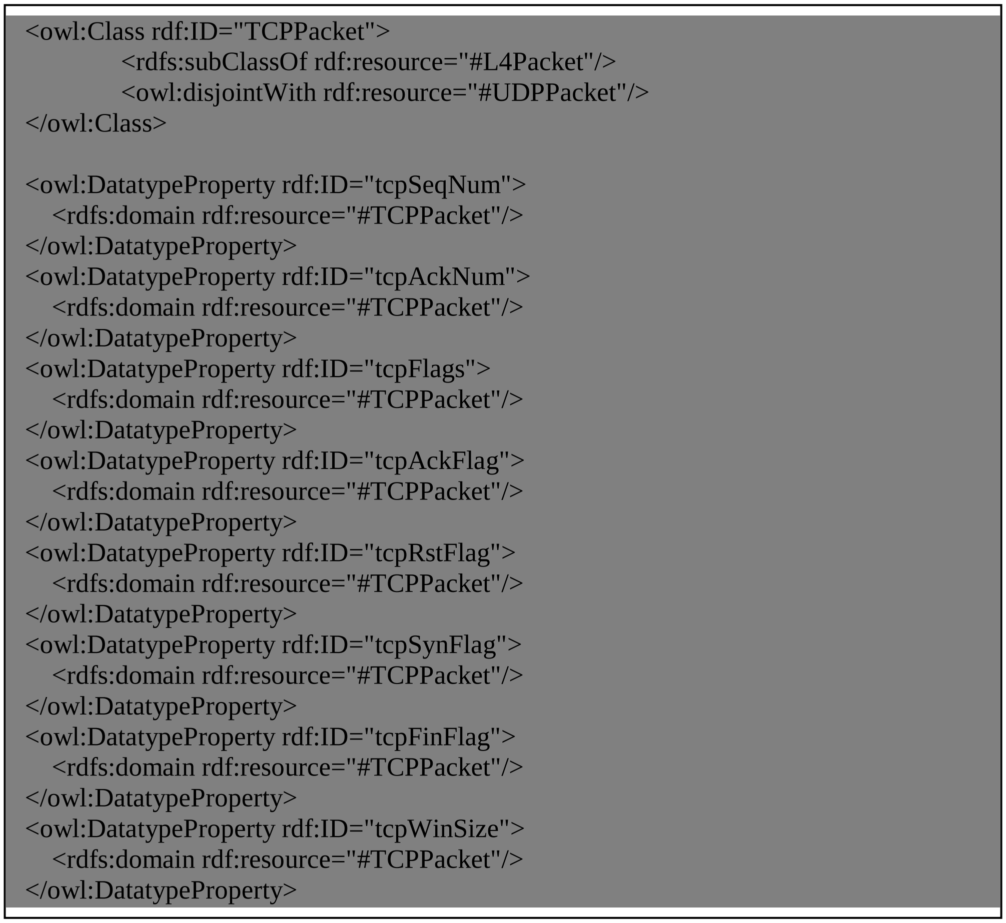


Figure 6.3: The traffic ontology [97].



```
<owl:Class rdf:ID="TCPPacket">
            <rdfs:subClassOf rdf:resource="#L4Packet"/>
            <owl:disjointWith rdf:resource="#UDPPacket"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="tcpSeqNum">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpAckNum">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpFlags">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpAckFlag">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpRstFlag">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpSynFlag">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpFinFlag">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpWinSize">
   <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
```

Figure 6.4: OWL code for *TCPPacket* in the traffic ontology.

From these basic packet instances, other instances are created in the knowledge base through the use of inference, which is performed by a reasoner. Again, consider the *TCPPacket* example. As seen in Fig. 6.3, the *TCPPacket* class is a subclass of the *L4Packet* class. When a *TCPPacket* instances is created, the reasoner will use inference to create an instance in the *L4Packet* class because of the subclass relation. The reasoner will continue to traverse up the class tree, creating instances in the parent classes. In this example, for every instance created in the *TCPPacket* class, instances are also created in the following classes: *L4Packet*, *IPPacket*, *L2Packet*, and *Packet*.

Based on instance properties, ontology constructs, and inference rules, packet collection instances are created. These instances represent groups of similar packets. For example, a Mask packet is an ICMP packet requesting the netmask value of the queried node. This type of ICMP packet is identified by a type value of 17. It is used as an information-gathering step in some complex attacks. The specification of this packet type in OWL is accomplished by obtaining all *ICMPPacket* instances with a restriction on the value of the *icmpType* property. This is done by using the *intersetionOf* construct and a property restriction. The OWL code for a Mask packet is provided in Fig 6.5.

```
<owl:Class rdf:ID="MaskPacket">
        <rdfs:comment>
          MaskPacket are ICMPPackets with ICMPtype of 17 (netmask request)
        </rdfs:comment>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#ICMPPacket"/>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#icmpType"/>
                        <owl:hasValue  rdf:datatype="&xsd;integer">17</owl:hasValue>
                </owl:Restriction>
        </owl:intersectionOf>
 </owl:Class>
```

Figure 6.5: OWL code for an ICMP netmask packet type.

As another example of the packet collection, consider the Ping of Death attack. This attack sends a large-sized ping packet to a host causing a buffer overflow at that host (the target machine). This attack uses a ping packet, which is an ICMP packet with a type of 8, with a packet length of 65535. OWL uses the *intersectionOf* construct with two property restrictions, one for the *icmpType* property and one for the *packetLen* property.

The *Stream* hierarchy in the traffic ontology is used to maintain information about past and present streams in the network. Instances are created for connection-oriented protocol streams, such as TCP, and non-connection-oriented protocol streams, such as UDP and ARP. This information is important to maintain to assist with the detection of attacks that modify address information, such as spoofing attacks.

When a host sends an ARP request and another host sends a response, considerable useful information is obtained. The source and destination MAC and IP addresses are learned. An instance in the *IPStream* class is created for this ARP communication containing the learned address information. If an attacker conducts an IP spoof against one of these hosts, the corresponding MAC address will differ from the one in the knowledge base. This leads to the detection of a possible IP spoof attack.

The other part of the traffic ontology is the alerts generated by Snort. The raw network traffic is run through Snort and an alert output file is created consisting of the alerts generated by Snort. This leverages an existing IDS to identify some of the simple attacks. For each alert generated by Snort, an instance is added to the knowledge base. Fig. 6.6 illustrates the part of the traffic ontology used for alerts. These instances are used by the attack ontology to identify the occurrence of specific attack elements.

6.3.2 Attack Representation

The attack ontology is used to maintain information about simple attacks. The attack data is obtained by using inference through ontology constructs and rules. Based on traffic instances created by the traffic subsystem, instances are added to the knowledge base using the attack ontology.

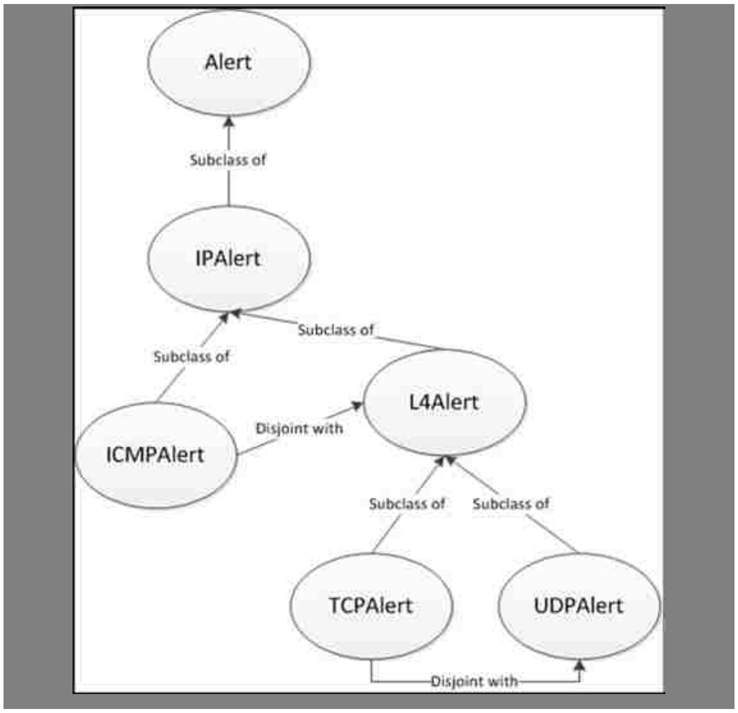
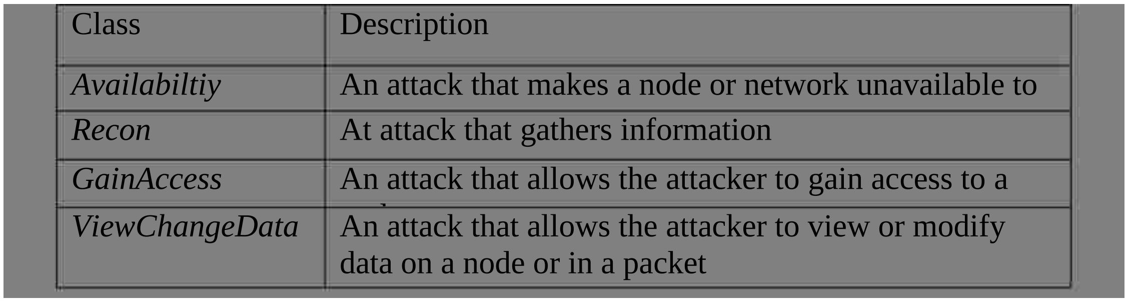


Figure 6.6: The alert part of the traffic ontology.

The primary class is the *Attack* class, which maintains much of the information about all types of attacks, such as the description of the attack, begin and end date and time, source IP address, and target IP address. There are four main classes of the Attack class. These classes are described in Table 6.1 and illustrated in Fig. 6.7.

Table 6.1: Main Classes of the Attack ontology.



Class	Description
Availabiltiy	An attack that makes a node or network unavailable to
Recon	At attack that gathers information
GainAccess	An attack that allows the attacker to gain access to a
ViewChangeData	An attack that allows the attacker to view or modify data on a node or in a packet

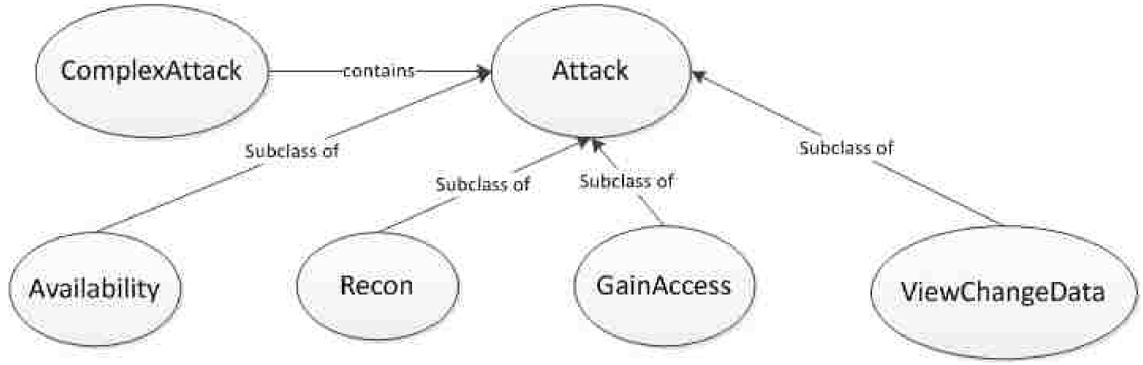


Figure 6.7: Main classes of the attack ontology.

To illustrate how the attack ontology follows from these main classes, one branch of the ontology hierarchy is shown in Fig. 6.8. This figure shows the various classes in the *Availability* branch of the ontology. There are two primary techniques an attacker will use to make a node or network unavailable. These two ways are a denial of service or spoofing attack. Each of these corresponds to a subclass of the *Availability* class and has several subclasses of their own.

One leaf node of the denial of service (*DoS*) hierarchy is *PingFlood*. This DoS attack uses many ping packets to flood a node or network consuming the resources and leaving no resources for other users. An instance of the *PingFlood* class, as well as the other flood nodes, is created from the *PacketCollection* instances in the traffic ontology. For each unique target IP address in the *PacketCollection* class, an instance is created in the *PingFlood* class, including the number of occurrences in the *PacketCollection* class for that target IP address. This frequency of occurrences is used when determining if an attack occurred. An attack occurs if the frequency is above a threshold value. For the purposes of this research, these values have been selected, rather than computed. Determination of the optimal threshold value will be addressed in future work.

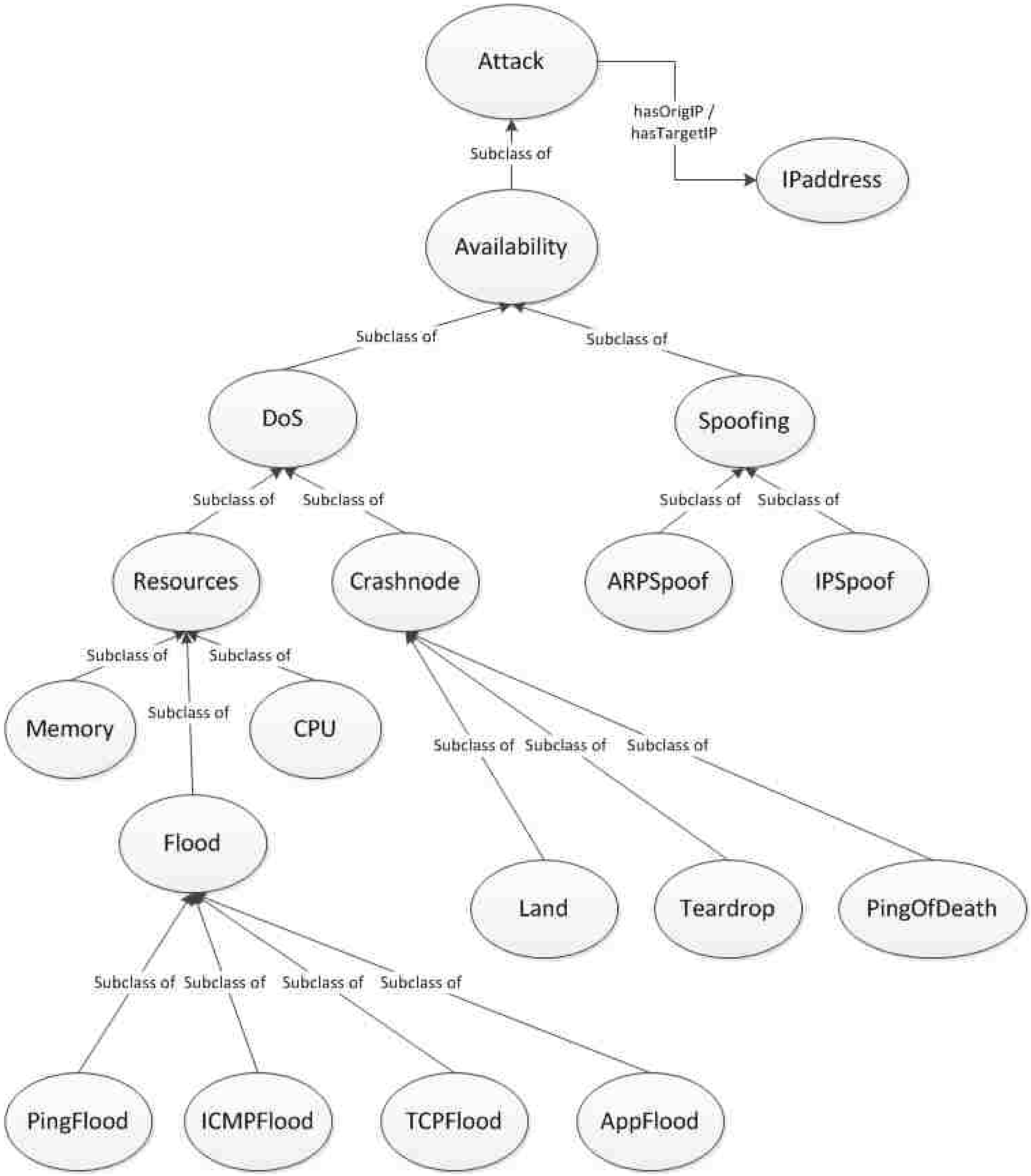


Figure 6.8: The availability branch of the attack ontology [97].

The *SimpleAttack* class is used to identify all occurrences of simple attacks. It is used to easily relay this attack information to the network manager. The instances in this class are the union of all instances in the four main attack classes (*Availability*, *Recon*, *GainAccess*, and *ViewChangeData*). The OWL code for collecting all *SimpleAttack* instances uses the *unionOf* construct and is shown in Fig. 6.9.

```
<owl:Class rdf:ID="SimpleAttack">
   <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#Availability"/>
     <owl:Class rdf:about="#Recon"/>
     <owl:Class rdf:about="#GainAccess"/>
     <owl:Class rdf:about="#ViewChangeData"/>
   </owl:unionOf>
</owl:Class>
```

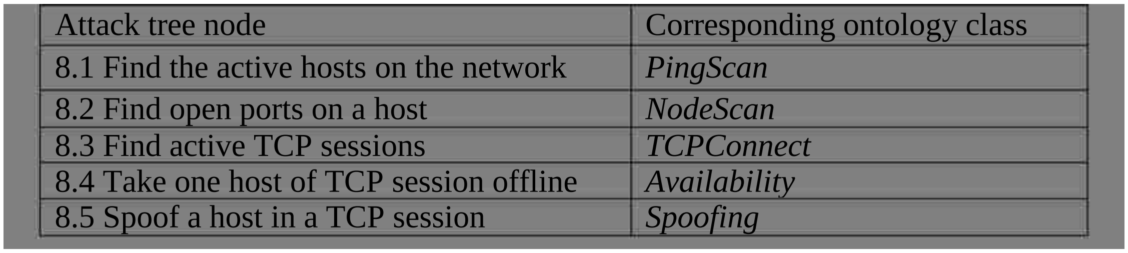
Figure 6.9: OWL code for the *SimpleAttack* class.

6.3.3 Complex Attack Representation

From the attack instances, complex attacks are identified. This is done by inferring the existence of attack elements for specific occurrences of complex attacks. The complex attack ontology, see Fig. 6.10, has instances created when the simple attack instances are created, and the ontology infers the parent instances in the complex attacks. When an instance is created in the root class of the complex attack ontology, it indicates that a complex attack occurred and the network manager is alerted.

The complex attack ontology was designed from the generic attack trees. Consider the generic attack tree in Fig. 6.11 illustrating a hijacking attack. There are five child nodes of the root node in the attack tree. Each of these nodes corresponds to a child node of the *Hijacking* class in the complex attack ontology (see Table 6.2).

Table 6.2: Attack Tree Nodes Link to Complex Attack Ontology Classes.



Attack tree node	Corresponding ontology class
8.1 Find the active hosts on the network	*PingScan*
8.2 Find open ports on a host	*NodeScan*
8.3 Find active TCP sessions	*TCPConnect*
8.4 Take one host of TCP session offline	*Availability*
8.5 Spoof a host in a TCP session	*Spoofing*

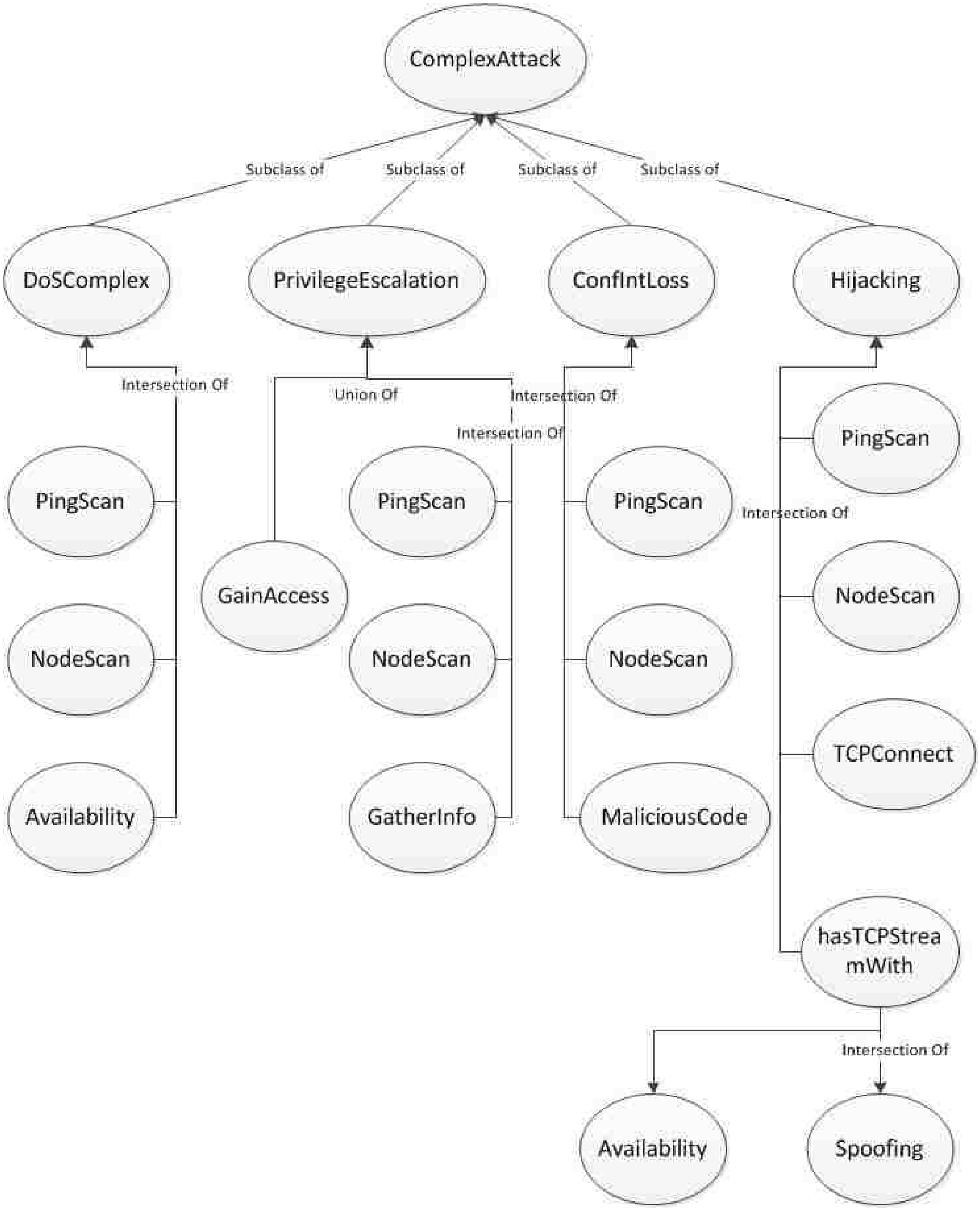


Figure 6.10: Complex attack portion of the attack ontology.

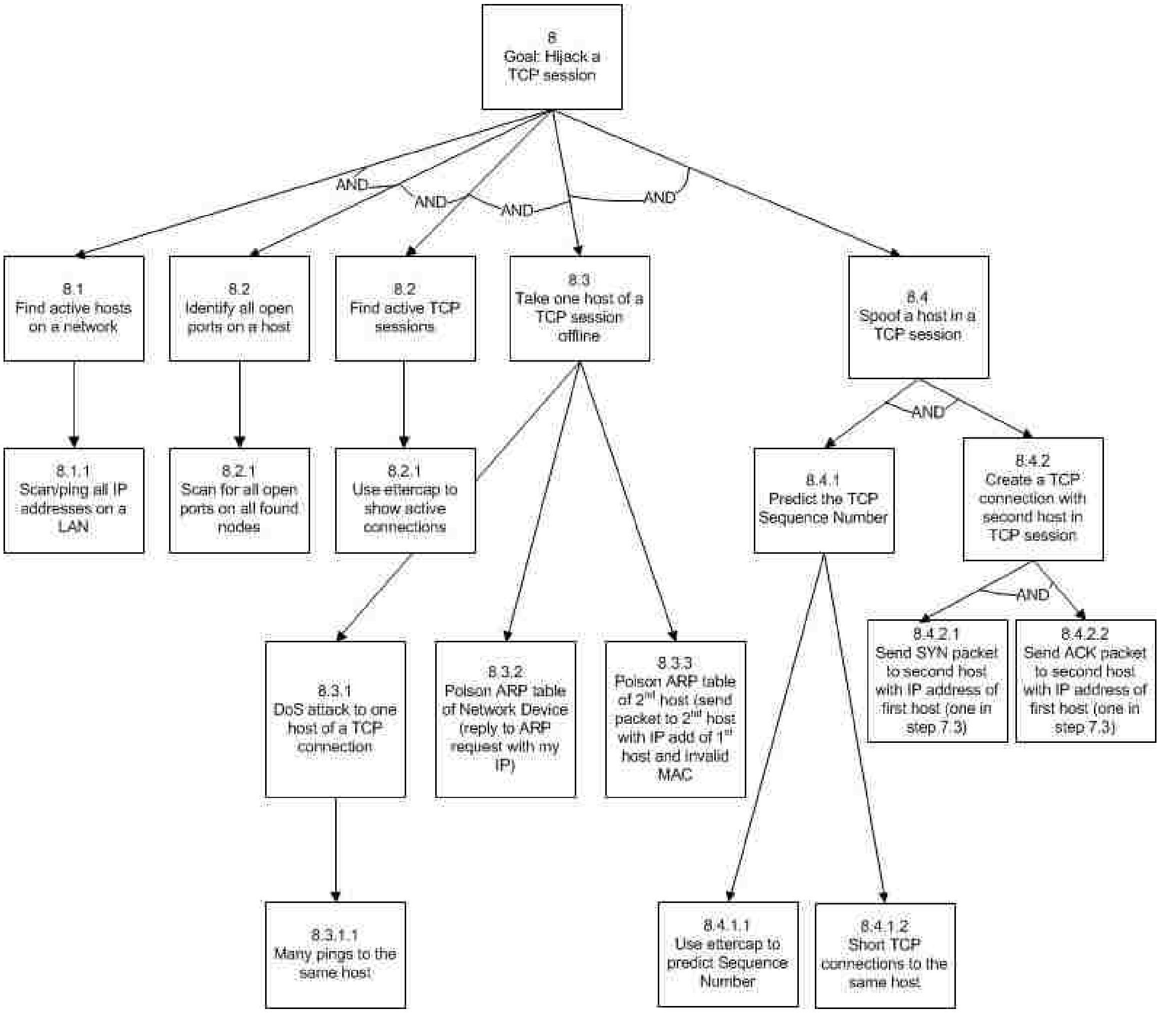

8
Goal: Hijack a TCP session
AND
AND
AND
AND
8.1
Find active hosts on a network
8.2
Identify all open ports on a host
8.2
Find active TCP sessions
8.3
Take one host of a TCP session offline
8.4
Spoof a host in a TCP session
AND
8.1.1
Scanvping all IP addresses on a LAN
8.2.1
Scan for all open ports on all found nodes
8.2.1
Use ettercap to show active connections
8.4.1
Predict the TCP Sequence Number
8.4.2
Create a TCP connection with second host in TCP session
AND
8.4.2.1
Send SYN packet to second host with IP address of first host (one in step 7.3)
8.4.2.2
Send ACK packet to second host with IP address of first host (one in step 7.3)
8.3.1
DoS attack to one host of a TCP connection
8.3.2
Poison ARP table of Network Device (reply to ARP request with my IP)
8.3.3
Poison ARP table of 2nd host (send packet to 2nd host with IP add of 1st host and invalid MAC
8.3.1.1
Many pings to the same host
8.4.1.1
Use ettercap to predict Sequence Number
8.4.1.2
Short TCP connections to the same host

Figure 6.11: An attack tree example.

These children nodes are not part of the complex attack ontology; they represent simple attacks and are part of the simple attack ontology. It is important to note that a hijacking attack is actually conducted against two target hosts, the two hosts in an established TCP connection. First, the attack will identify an active host (ping scan), an active TCP connection on that host (node scan), and then predict the TCP sequence number for that TCP connection (TCP connect attack). The attacker then targets the other host in the TCP connection to make it unavailable to respond to requests from the first host (DoS attack) and spoof it’s IP address (spoofing attack). The complex intersection of these five classes indicates the occurrence of a complex hijacking attack. The OWL code for the *Hijacking* class is shown in Fig. 6.12.

```
<owl:Class rdf:ID="Hijacking">
  <rdfs:comment>
            A complex Hijacking attack is a Ping scan, Node
            scan, TCP Scan, Availability and Spoofing attack
  </rdfs:comment>

  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&traffic;NWaddressScanned"/>
      <rdf:Description rdf:about="&traffic;IPaddress"/>
      <owl:Restriction>
            <owl:onProperty rdf:resource="&attack;wasAttacked"/>
            <owl:someValuesFrom rdf:resource="&attack;NodeScan"/>
      </owl:Restriction>
      <owl:Restriction>
            <owl:onProperty rdf:resource="&attack;wasAttacked"/>
            <owl:someValuesFrom rdf:resource="&attack;TCPConnect"/>
      </owl:Restriction>
      <owl:Restriction>
            <owl:onProperty rdf:resource="&traffic;hasTCPStreamWith"/>
            <owl:someValuesFrom>
             <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&traffic;IPaddress"/>
                <owl:Restriction>
                  <owl:onProperty rdf:resource="&attack;wasAttacked"/>
                  <owl:someValuesFrom
                              rdf:resource="&attack;Availability"/>
                </owl:Restriction>
                <owl:Restriction>
                  <owl:onProperty rdf:resource="&attack;wasAttacked"/>
                  <owl:someValuesFrom rdf:resource="&attack;Spoofing"/>
                </owl:Restriction>
              </owl:intersectionOf>
             </owl:Class>
            </owl:someValuesFrom>
      </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Figure 6.12: OWL code for the *Hijacking* class.

The code for the *Hijacking* class returns the instances for hosts with all of the following events:

- Host A' network was scanned
- A node scan attack was performed against host A
- A TCP connect attack was performed against host A
- An availability attack was performed against the other host in the TCP connection, host B
- A spoofing attack was performed against host B

The OWL code will identify all instances that meet these criteria. This is done by using the *intersectionOf* all *IPaddress* instances that have their network scanned (*NWaddressScanned*), had a *NodeScan* against them, had a *TCPConnect* scan against them, and had a TCP connection with (*hasTCPStreamWith*) another host. This second host had two attacks against it, an *Availability* attack and a *Spoofing* attack. All IP addresses that meet these criteria are identified as a target of a hijacking attack in TRIDSO.

This example will follow a complex attack through the entire ontology as an illustration of how all instances are created. The example is for a complex denial of service attack. The following are an example of the steps an attacker may take when conducting a complex denial of service attack:

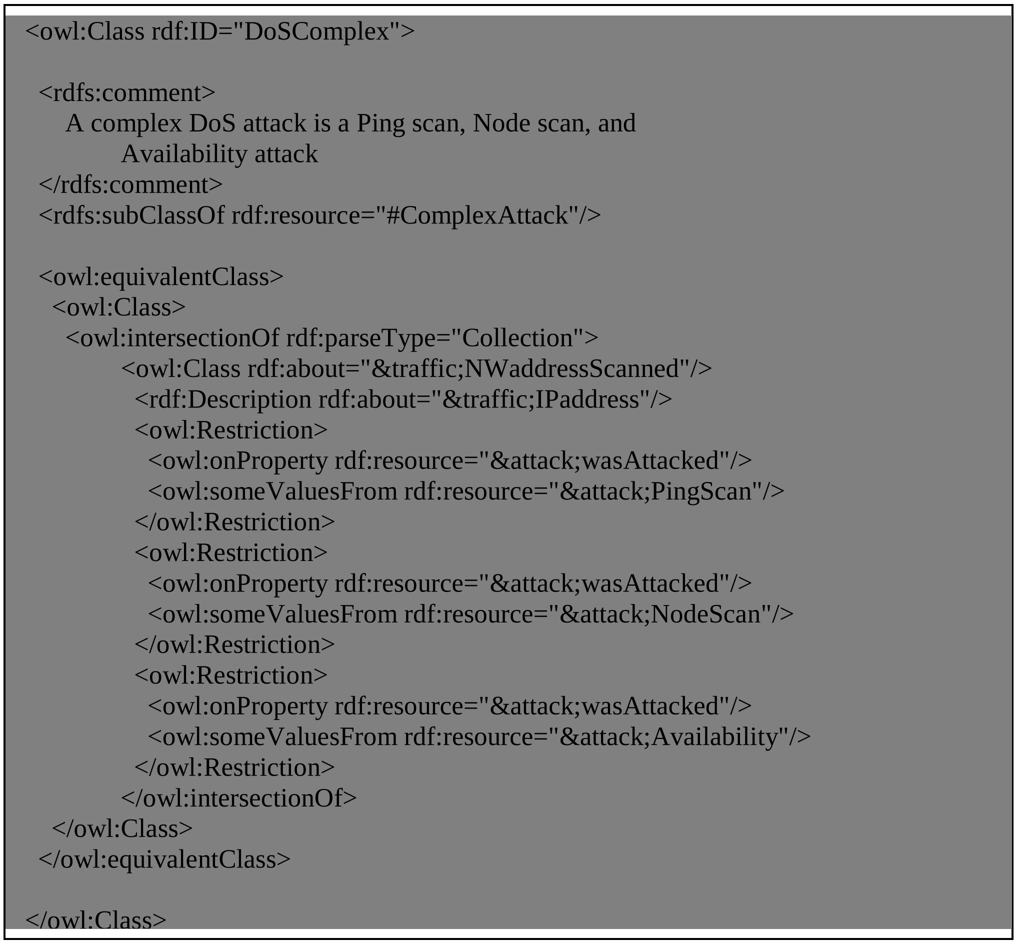
1. Scan all nodes on a network to see which nodes respond indicating they are active.

2. Scan all ports on an active node on the network to see which ports are active and listening for requests.
3. Take a node off-line by sending many ping packets to it making it unavailable to users.

The first step includes the attacker sending a ping packet to every IP address on a network. A response indicates the node is active. These packets are added to the traffic ontology as *ICMPPacket* instances since ping uses ICMP. From these instances, it is determined through a rule (rules are explained in chapter 7) that a *PingScan* occurred and an instance is added to the *PingScan* class in the attack ontology. A similar sequence happens for the *NodeScan* class for the second step. The third step results in the creation of a *PingFlood* instance.

Inference, through taxonomic relationships, specifically subclass, causes an instance to occur in the following classes: *Flood*, *Resources*, *DoS*, and *Availability*. Now, there exist instances in the *PingScan*, *NodeScan*, and *Availability* classes in the attack ontology. Because of the definition of the *DoSComplex* class, shown in Fig. 6.13, an instance is created in that class, indicating that a complex denial of service attack occurred.

A denial of service complex attack may only consist of the first and third steps above; it is possible to launch the availability attack against a node or the network without knowing all open ports on a node(s). In this case, the *DoSComplex* class only consists of the intersection of the *PingScan* and *Availability* instances. This OWL code is the same as in Fig. 6.13 except the restriction for the *NodeScan* is removed. The full *DoSComplex* class definition is then the union of these two class definitions.



```
<owl:Class rdf:ID="DoSComplex">

  <rdfs:comment>
     A complex DoS attack is a Ping scan, Node scan, and
           Availability attack
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#ComplexAttack"/>

  <owl:equivalentClass>
   <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
           <owl:Class rdf:about="&traffic;NWaddressScanned"/>
             <rdf:Description rdf:about="&traffic;IPaddress"/>
             <owl:Restriction>
               <owl:onProperty rdf:resource="&attack;wasAttacked"/>
               <owl:someValuesFrom rdf:resource="&attack;PingScan"/>
             </owl:Restriction>
             <owl:Restriction>
               <owl:onProperty rdf:resource="&attack;wasAttacked"/>
               <owl:someValuesFrom rdf:resource="&attack;NodeScan"/>
             </owl:Restriction>
             <owl:Restriction>
               <owl:onProperty rdf:resource="&attack;wasAttacked"/>
               <owl:someValuesFrom rdf:resource="&attack;Availability"/>
             </owl:Restriction>
           </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>

</owl:Class>
```

Figure 6.13: OWL code for the *DoSComplex* class.

6.4 Chapter Summary

The formal representation presented in this chapter provides a high-level abstraction of the network activity. It bridges the gap between the raw data and how humans view sophisticated attacks. It eliminates the need to have specific patterns to match against to detect the occurrence of an attack.

A RIDS can be used to identify complex attacks and attack attempts. The RIDS developed in this research (Traffic-based Reasoning Intrusion Detection System using Ontology, TRIDSO) bases the attack detection on all network traffic, not just certain systems or known vulnerabilities. It will detect generic attacks and attack attempts, possibly even zero-day attacks, by analyzing specific attack elements and incorporating these elements into attack trees.

This IDS, unlike its predecessors, uses ontological technology to reason about traffic and what specific packets may represent in the context of undesirable traffic. Some advanced ontology constructs, such as subclasses, unions, and intersections, allow inference within the ontology. The use of reasoning will allow TRIDSO to detect more attacks and attack attempts than traditional IDSs, as evidence by the evaluation of TRIDSO explained in the next chapter.

Another advantage of TRIDSO is that the initial versions of the ontologies can be augmented over time due to the flexibility and portability of ontology. This may include the addition of new attack representations, allowing the detection of all attacks and attack attempts. Ultimately the TRIDSO ontology may be extended by many different network managers.

Chapter 7 **Complex Attack Reasoning and Recognition**

Intrusion Detection Systems (IDSs) are utilized to detect attacks against host and networks. IDSs are one type of application that may benefit from the many advantages provided by ontology. Some of the advantages provided by ontology include inference, advanced semantic expressiveness, flexibility, and portability. While these advantages are beneficial to RIDSs, there exist some shortcomings of ontology. Some of the requirements of the RIDS developed in this research not supported by ontology are the ability to select instances based on ranges of values for a specified field, selecting instances that have a field that is optional, performing aggregate operations on values to obtain results, and selecting instances based on regular expression matching. Some of the useful aggregate operations are finding the minimum value from a set of instances, the maximum value from a set of instances, or counting the number of matching instances. To satisfy these requirements in the RIDS, SPARQL [17], a query language for use with ontology applications, is used.

7.1 A Set of Heuristics for Complex Attack Identification

A set of high-level conceptual heuristics is developed, using SPARQL, to process the declarative representation of captured network data to aid in detecting complex attacks and attack attempts. The set of heuristics is used to perform some advanced processing of the instances in the knowledge base to create additional instances. Specifically, SPARQL is used to create instances for packet collections, packet streams, and simple attacks. Instances are created in the knowledge base.

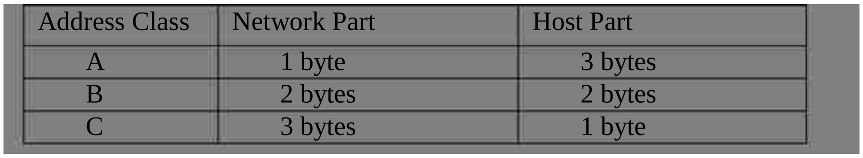
One advantage of SPARQL is its flexibility. The rules developed are generic, allowing for the identification of instances for general types of attacks and not occurrences of specific attacks.

The set of rules developed are also extensible. The ontology definition contains much of the necessary information for the attacks. For this reason, it is not difficult to create a new rule for a newly identified type of attack. This allows other researchers to add to the set of rules allowing for the detection of additional attacks.

SPARQL rules are used to create instances in the *PacketCollection* class. These instances are created for groups of instances in the traffic ontology. *PacketCollection* instances exist for various types of floods and scans. A flood is a group of packets that are generated in quantity to utilize a lot of resources. The most common type of flood is a ping flood, which is a large amount of ping packets sent to consume bandwidth. The ping packets may be sent to one specific host or multiple hosts in a network. Other types of floods are ICMP, TCP and application. These floods are similar to ping floods but use other packet types. The SPARQL rules for the prototype system are in Appendix C.

Finding ping scans to multiple nodes on a network is fairly complex in SPARQL because it requires finding the network address corresponding to the IP address of each node. An IP address is split into two main parts, the network part and the host part. The parts vary in size (the number of bytes) depending on the class of the IP address. Table 7.1 shows the number of bytes corresponding to the network and host part of the IP address for the three classes of IP addresses used for hosts in a network. The network address consists of the network number part of the IP address and zero for each host part of the IP address.

Table 7.1: Network Address Compilation.



Address Class	Network Part	Host Part
A	1 byte	3 bytes
B	2 bytes	2 bytes
C	3 bytes	1 byte

The rule to find ping scans to a class B network, shown in Fig. 7.1, must look for packets sent to the multiple IP addresses on the same network. This requires finding the network address for each IP address for all ping packets in the traffic ontology and then determining the number of pings sent to nodes on the same network. This requires more complex matching in the rule because it involves instances from multiple classes; the *PingPacket* class to find all ping packets and the *IPaddress* class to find the network address for each target IP address in Ping packets. The rule also requires the concatenation of fields to obtain the network IP address for the Ping packet.

Scans gather information about the network or nodes on the network. The two common scans are ping scan and port scan. A ping scan is conducted to find nodes on the network that are active. It consists of sending a ping packet to each possible IP address to see which nodes respond, indicating an active node. After finding active nodes, it is common to run a port scan on each active node. A port scan is performed to find which services are active on a specific node. Now the attacker knows possible points of attack (open ports on active nodes are possible points of attack). Finding port scans to one node is fairly simple using SPARQL. The rule, shown in Fig. 7.2, looks for packets sent to multiple ports on the same IP address.

```
PREFIX traffic: <traffic.owl#>
PREFIX attack: <attack.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT  {
        _:a rdf:type attack:PacketCollection;
             attack:beginDate ?beginDateTime;
             attack:endDate ?endDateTime;
             attack:pcType traffic:PingScanType;
             attack:hasTargetIP ?nwadd;
             attack:pcFrequency ?cnt .
 }   WHERE { {
        SELECT ?nwadd ?IPoctet1 ?IPoctet2
                     (MIN(?dateTime) as ?beginDateTime)
                     (MAX (?dateTime) as ?endDateTime)
                     (count(?nwadd) as ?cnt)
        {
          SELECT DISTINCT ?packet1 ?ipadd1 ?IPoctet1 ?IPoctet2
                         ?IPoctet3a ?IPoctet4a ?nwadd ?dateTime
          {
            ?packet1 rdf:type traffic:PingPacket;
                    traffic:hasDestIP ?ipadd1;
                    traffic:dateTime ?dateTime .
            ?ipadd1  rdf:type traffic:IPaddress;
                    traffic:IPoctet1 ?IPoctet1;
                    traffic:IPoctet2 ?IPoctet2;
                    traffic:IPoctet3 ?IPoctet3a;
                    traffic:IPoctet4 ?IPoctet4a .
            ?nwadd apf:concat (?IPoctet1 "." ?IPoctet2 ".0.0")
            {
             SELECT DISTINCT ?packet2 ?ipadd2 ?IPoctet1 ?IPoctet2
                           ?IPoctet3b ?IPoctet4b ?nwadd2
              {
               ?packet2 rdf:type traffic:PingPacket;
                       traffic:hasDestIP ?ipadd2;
                       traffic:dateTime ?dateTime2 .
               ?ipadd2  rdf:type traffic:IPaddress;
                       traffic:IPoctet1 ?IPoctet1;
                       traffic:IPoctet2 ?IPoctet2;
                       traffic:IPoctet3 ?IPoctet3b;
                       traffic:IPoctet4 ?IPoctet4b .
               ?nwadd2 apf:concat (?IPoctet1 "." ?IPoctet2 ".0.0")
          }  }
          FILTER ( ( ?packet1 != ?packet2 ) &&
                  ( ?IPoctet1 >= 128 ) &&
                  ( ?IPoctet1 <= 191 ) ) .
        }   }
        GROUP BY ?nwadd ?IPoctet1 ?IPoctet2
}  }
```

Figure 7.1: A SPARQL rule to describe a class B network ping scan.

```
PREFIX traffic: <traffic.owl#>
PREFIX attack: <attack.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT
{
    _:a  rdf:type  attack:PacketCollection;
         attack:beginDate  ?beginDateTime;
         attack:endDate  ?endDateTime;
         attack:pcType traffic:PortScanType;
         attack:hasTargetIP ?destIP;
         attack:pcFrequency ?cnt .
}   WHERE { {
    SELECT DISTINCT ?packet1 ?destIP
                  (MIN(?dateTime) as ?beginDateTime)
                  (MAX (?dateTime) as ?endDateTime)
                  (count(?destIP) as ?cnt)
      {
        ?packet1 rdf:type traffic:L4Packet;
                   traffic:dateTime ?dateTime;
                   traffic:hasDestIP ?destIP;
                   traffic:l4DestPort ?l4DestPort1 .
        {
          SELECT ?packet2 ?destIP ?l4DestPort2 ?dateTime2
          {
           ?packet2  rdf:type traffic:L4Packet;
                       traffic:dateTime ?dateTime2;
                       traffic:hasDestIP ?destIP;
                       traffic:l4DestPort ?l4DestPort2 .
          }
          GROUP BY ?destIP
        }
        FILTER ( ( ?packet1 != ?packet2) &&
                   ( ?l4DestPort1 != ?l4DestPort2 ) ) .
      }
     GROUP BY ?destIP
     HAVING (count(?destIP) > 0)
}   }
```

Figure 7.2: A SPARQL rule to describe a node port scan.

TrafficStream instances are also created by using SPARQL rules. These instances are created containing information for source and destination nodes for all TCP, UDP, ICMP, layer 3, and ARP packets sent. This information is used to identify possible spoof attacks; attacks where a third node pretends to be one of the original nodes in the communication.

The SPARQL rule for creating a *TrafficStream* instance for a TCP connection is shown in Fig. 7.3.

```
PREFIX traffic: <traffic.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT
 {
  ?stream rdf:type traffic:TCPStream;
              traffic:protocol \"TCP\";
              traffic:startTime ?dateTime;
              traffic:endTime ?dateTime;
              traffic:hasNode1MAC ?srcMAC;
              traffic:hasNode2MAC ?destMAC;
              traffic:hasNode1IP ?srcIP;
              traffic:hasNode2IP ?destIP;
              traffic:node1Port ?l4SrcPort;
              traffic:node2Port ?l4DestPort .
}
WHERE  { {
     SELECT DISTINCT ?packet ?dateTime ?srcMAC ?destMAC
                ?srcIP ?destIP ?l4SrcPort ?l4DestPort
        {
          ?packet rdf:type traffic:TCPPacket;
                       traffic:dateTime ?dateTime;
                       traffic:hasSrcMAC ?srcMAC;
                       traffic:hasDestMAC ?destMAC;
                       traffic:hasSrcIP ?srcIP;
                       traffic:hasDestIP ?destIP;
                       traffic:l4SrcPort ?l4SrcPort;
                       traffic:l4DestPort ?l4DestPort .
      }   }
     LET (?stream := ?packet) .
}
```

Figure 7.3: A SPARQL rule to describe a *TCPStream*.

SPARQL rules are also used to create instances for simple attacks. Snort identifies some simple attacks. The information about these attacks is useful and is utilized by this research. Recall that instances exist in the traffic ontology for all Snort alerts generated. These instances are matched against regular expressions using SPARQL rules to find occurrences of specific simple attacks. For example, a SPARQL rule is used to find all attacks where an attacker gained root access on a node. The code for this SPARQL rule is shown in Fig. 7.4.

```
PREFIX traffic: <traffic.owl#>
PREFIX attack: <attack.owl#>
INSERT
{
        ?attack rdf:type attack: “AdminPG ";
                    attack:attBeginDate ?aDateTime;
                    attack:attEndDate ?aDateTime;
                    attack:description ?aDesc;
                    attack:targetAddress ?aDestIP .
}
WHERE  { {
      SELECT ?alert ?aDateTime ?aDesc ?aDestIP
        {
           ?alert rdf:type traffic:Alert;
                     traffic:aDateTime ?aDateTime;
                     traffic:aDescription ?aDesc;
                     traffic:aClassification ?aClassification .
          OPTIONAL { ?alert traffic:aDestIP ?aDestIP . } .
          FILTER REGEX(?aClassification, "Administrator Privilege Gain", "i") .
      }    }
     LET (?attack := ?alert) .
}
```

Figure 7.4: A SPARQL rule to describe a simple attack for gaining root access.

Instances for other simple attacks are also created using SPARQL rules. These instances include Land attacks, which are identified by the source IP address and port number being the same as the destination IP address and port number in a packet. Another rule exists to find possible ARP spoofs. This spoof occurs when an attacker modifies the MAC address of their host to match the MAC address of a target host. This type of attack is identified by a packet with an IP address associated with a different MAC address than previously observed.

7.2 Development of a Prototype System

A prototype system was developed [97] to show the feasibility of using the ontology and set of heuristics for detecting attacks. The prototype system developed is the Traffic-based Reasoning Intrusion Detection System with Ontology (TRIDSO). TRIDSO (see Fig. 7.5) consists of a variety of subsystems: traffic, attack, vulnerability, and device. Each subsystem consists of a variety of components, including an ontology definition file. TRIDSO provides data-driven reasoning; the reasoning and decisions are based on traffic data.

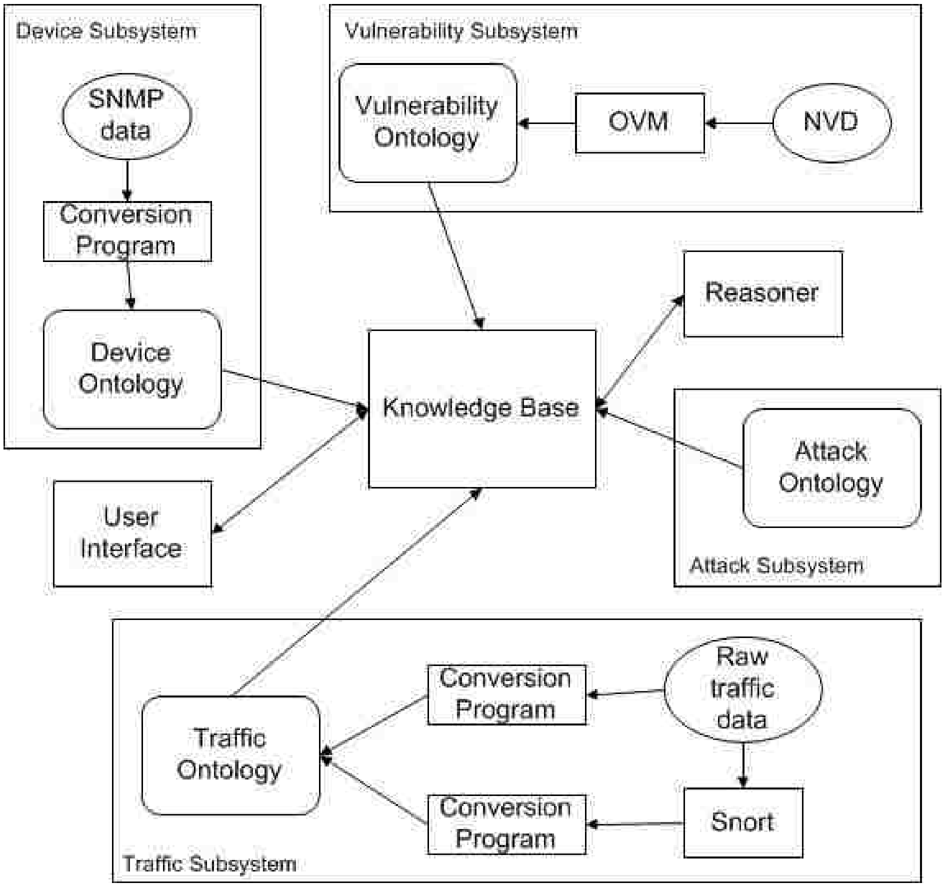
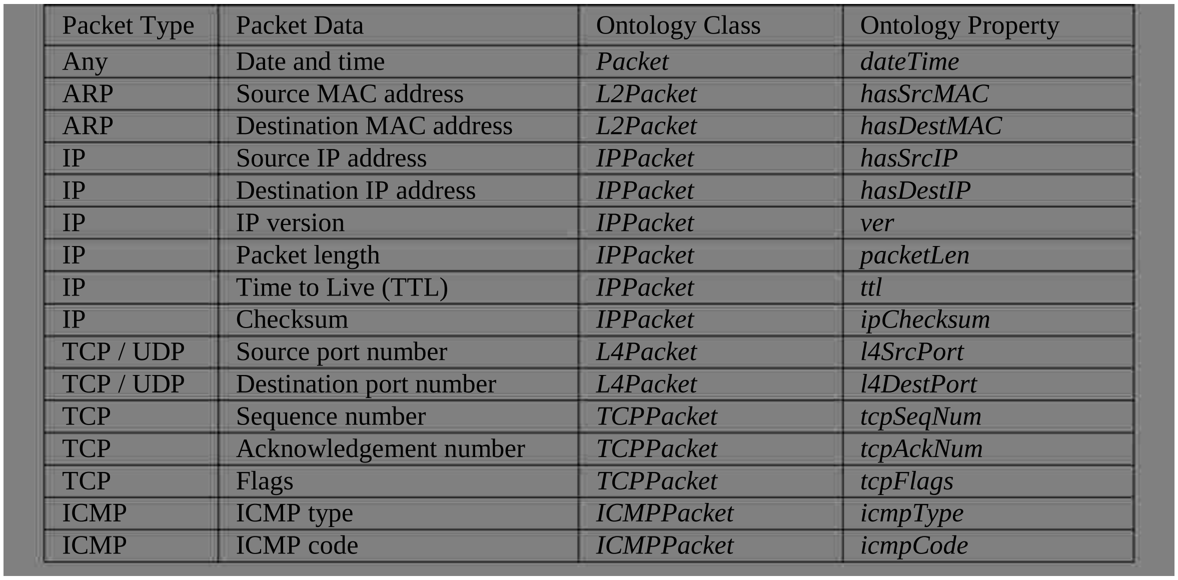


Figure 7.5: TRIDSO architecture [97].

TRIDSO was developed using Java and Jena [18], a framework for ontology applications. Jena was chosen primarily because it provides a leading implementation of SPARQL. This implementation includes support for SPARQL extensions, such as INSERT and *count*, which are necessary in the set of heuristics developed.

The traffic subsystem deals with raw network traffic data. Wireshark [98, 99] is used to capture all network traffic. A program converts this data to ontology instances in the traffic ontology. This conversion program reads through a tcpdump-formatted capture file. For each packet found, the type of packet is determined, such as TCP, UDP, IP or ARP, and the required data for that packet type is extracted. An instance is then created for each packet in the appropriate class. A sampling of the relationships between packet data and ontology properties is provided in Table 7.2.

Table 7.2: Relationship Between Packet Data and Ontology Property.

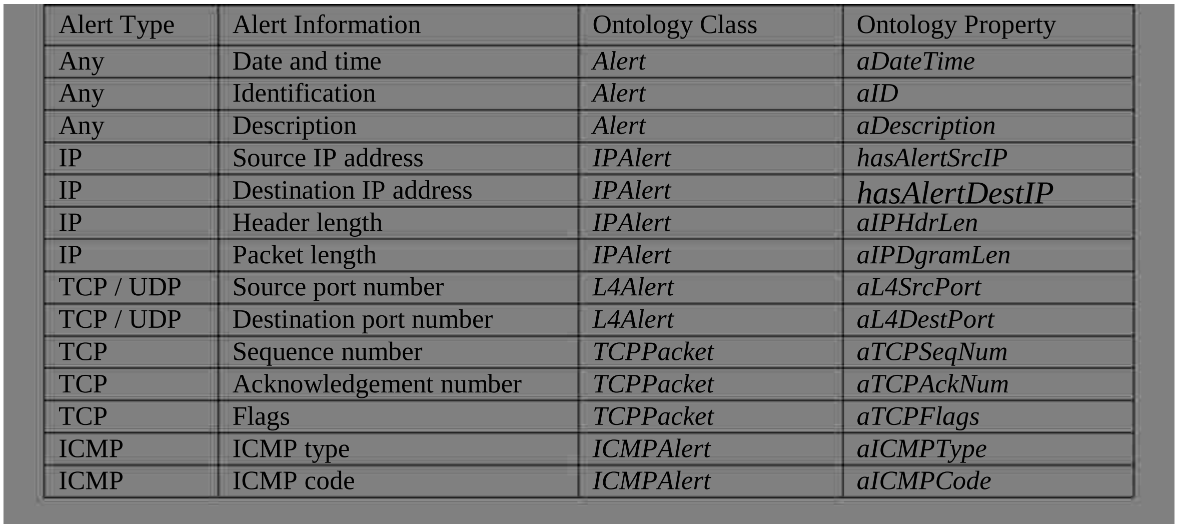


Packet Type	Packet Data	Ontology Class	Ontology Property
Any	Date and time	*Packet*	*dateTime*
ARP	Source MAC address	*L2Packet*	*hasSrcMAC*
ARP	Destination MAC address	*L2Packet*	*hasDestMAC*
IP	Source IP address	*IPPacket*	*hasSrcIP*
IP	Destination IP address	*IPPacket*	*hasDestIP*
IP	IP version	*IPPacket*	*ver*
IP	Packet length	*IPPacket*	*packetLen*
IP	Time to Live (TTL)	*IPPacket*	*ttl*
IP	Checksum	*IPPacket*	*ipChecksum*
TCP / UDP	Source port number	*L4Packet*	*l4SrcPort*
TCP / UDP	Destination port number	*L4Packet*	*l4DestPort*
TCP	Sequence number	*TCPPacket*	*tcpSeqNum*
TCP	Acknowledgement number	*TCPPacket*	*tcpAckNum*
TCP	Flags	*TCPPacket*	*tcpFlags*
ICMP	ICMP type	*ICMPPacket*	*icmpType*
ICMP	ICMP code	*ICMPPacket*	*icmpCode*

Instances are added to the knowledge base using the *createIndividual* function in the Jena library. The function used to add the properties for each instance depends on the property type. For datatype properties, two functions are used. To create an OWL literal value, *createTypedLiteral* is used and then the literal is added as the property value using *createLiteralStatement*. If the property is an object property, the object that is the value of the property must already exist in the knowledge base. If it does not, it is added as an instance. To create the actual statement relating the subject to the object for the object property, the function *createStatement* is used.

Data is also added to the knowledge base for alerts identified by Snort. Prior to running the RIDS, the tcpdump-formatted capture file is run through Snort, which generates an alert file. The alerts in the alert file are read by the RIDS, which creates appropriate instances in the alert classes. Table 7.3 lists some of the alert information from the alert file and their relationships with the ontology properties.

Table 7.3: Alert Information's Relationship with Ontology Property.



Alert Type	Alert Information	Ontology Class	Ontology Property
Any	Date and time	*Alert*	*aDateTime*
Any	Identification	*Alert*	*aID*
Any	Description	*Alert*	*aDescription*
IP	Source IP address	*IPAlert*	*hasAlertSrcIP*
IP	Destination IP address	*IPAlert*	*hasAlertDestIP*
IP	Header length	*IPAlert*	*aIPHdrLen*
IP	Packet length	*IPAlert*	*aIPDgramLen*
TCP / UDP	Source port number	*L4Alert*	*aL4SrcPort*
TCP / UDP	Destination port number	*L4Alert*	*aL4DestPort*
TCP	Sequence number	*TCPPacket*	*aTCPSeqNum*
TCP	Acknowledgement number	*TCPPacket*	*aTCPAckNum*
TCP	Flags	*TCPPacket*	*aTCPFlags*
ICMP	ICMP type	*ICMPAlert*	*aICMPType*
ICMP	ICMP code	*ICMPAlert*	*aICMPCode*

Raw network data is captured using Wireshark. Snort is run on the raw data to produce an alert file. The Wireshark capture file and alert file are processed by conversion programs and instances are added to the knowledge base. This flow of data for the traffic subsystem is illustrated in Fig. 7.6.

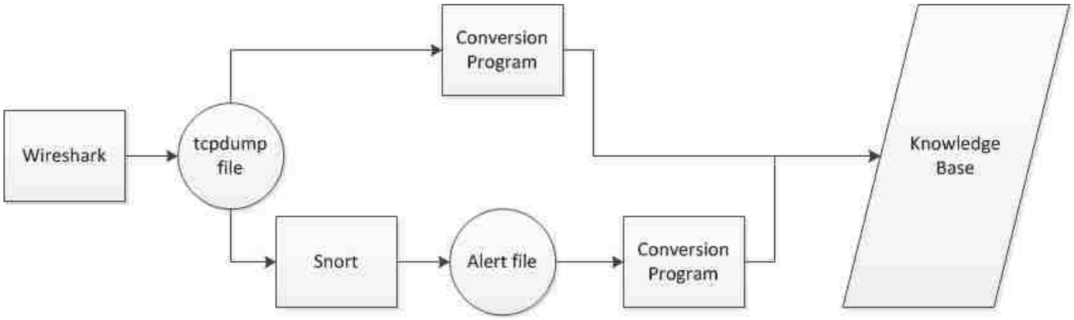


Figure 7.6: The data flow of the traffic subsystem.

The attack subsystem consists of an ontology that will hold attack data. There are actually two ontology definition files in the attack subsystem, the attack ontology and the complex attack ontology. Two files are used to simplify the maintenance of the ontology files. The attack ontology contains class definitions for all simple attacks. Complex attack classes are defined in the complex attack ontology.

The attack instances are created in a variety of methods. Some are added using SPARQL from traffic ontology instances. For example, scan and flood attack instances are created using SPARQL queries based on *PacketCollection* instances.

Some simple attacks are detected by Snort. Some of these are added as simple attacks in the knowledge base. Attacks detected by Snort that are to be added to the knowledge base are identified using regular expression matches in various alert instance properties. For instance, some Snort alerts indicate a malicious code type of attack. These are identified by finding alert instances with the following strings in the classification property (these are just some examples, there are more strings identifying a malicious code attack): *Decode of an RPC Query*, *Executable Code was Detected*, *A Suspicious*

String was Detected, *Access to a Potentially Vulnerable Web Application,* and *A System Call was Detected*.

The vulnerability subsystem manages the existing vulnerabilities. The ontology in this subsystem contains data about vulnerabilities. The development of this subsystem has not been completed and is left as future work. The data will be loaded into the ontology from existing sources, such as NIST's NVD, OVAL [65], or Snort rules. There is reason to believe that the NVD data can be obtained from OVM (Ontology for Vulnerability Management) [100], which is existing research that loads NVD data to an ontology. To determine vulnerabilities of hosts, a vulnerability scanner, such as nessus [101] or SSA Security System Analyzer [102], may be used.

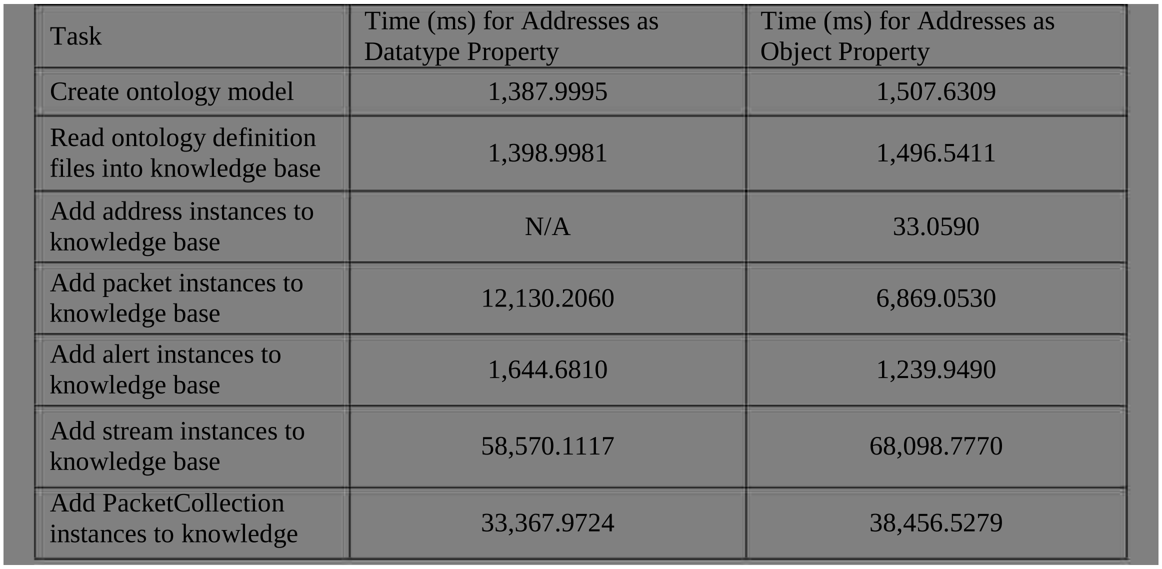
The device subsystem consists of the device ontology and a program to convert device data to ontology instances. The ontology contains classes representing the devices in the network and their characteristics. This data is retrieved from the devices using a standard management protocol, such as SNMP. After the device information is retrieved, instances are added to the knowledge base using the devices ontology. Initially, the devices are routers and switches. The device ontology has been developed for the HMNMS discussed in chapter 4. Future work will include incorporating this ontology into the device subsystem of TRIDSO.

7.2.1 Implementation Decisions

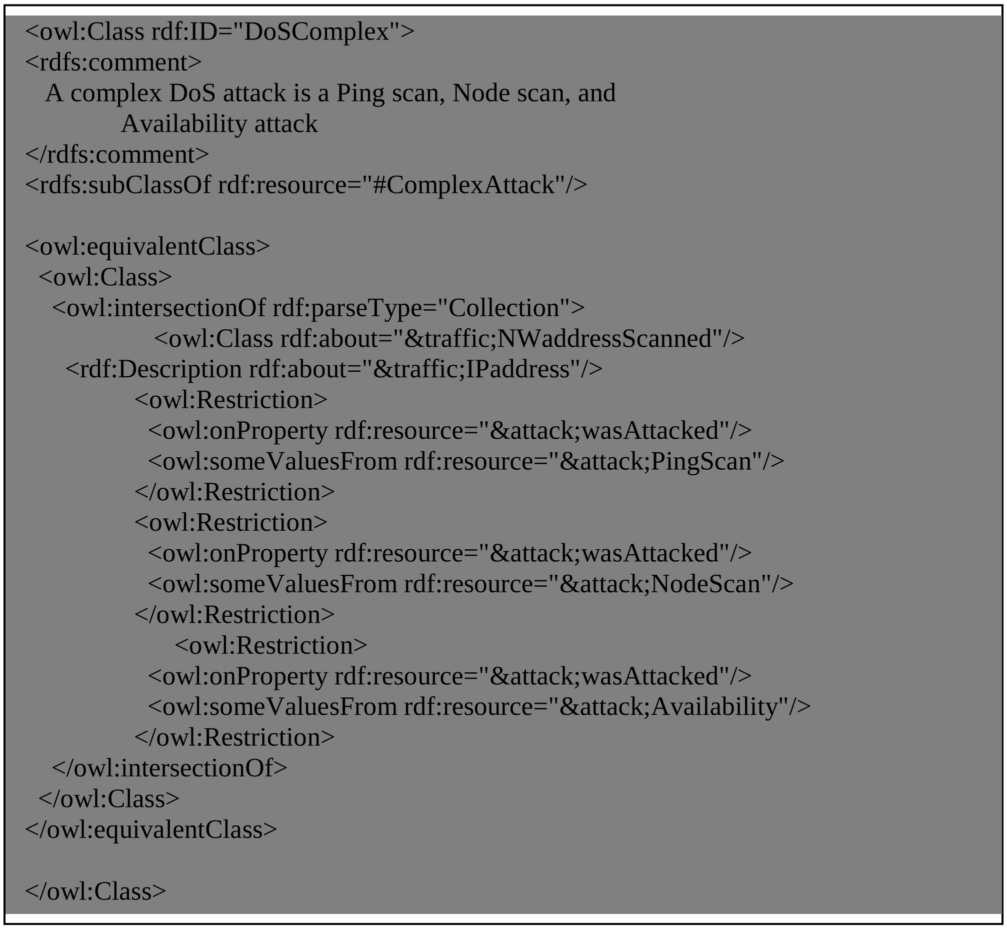
The design and development of TRIDSO included some implementation decisions. Decisions had to be made between datatype vs. object properties in OWL, using OWL vs. SPARQL, and using various SPARQL statements. For two of the implementation decisions, two different implementations were completed in the existing version of TRIDSO and trials were conducted against a test data file.

The first implementation decision was how to represent the nodes' addresses, both MAC and IP, in the ontology. Two options were considered, using a datatype property or an object property for the addresses. As seen from the results in Table 7.4, the implementation using the datatype property runs faster than the implementation using the object property; however, there are other advantages to using the object property. The primary advantage is the inference available when using the object property. Object properties can have inverse properties defined. This switches the subject and object in the triples. For example, if an attack is executed against a specified IP address, then that address is the object for the *hasTargetIP* property. The *wasAttacked* property is declared to be the *inverseOf* the *hasTargetIP* property. The IP address is now the subject and the attack element is the object of that property. This allows the ontology reason to automatically add instances to the knowledge base. This is beneficial when identifying complex attacks. For example, a *DoSComplex* attack can be identified by finding each IP address that was attacked using a *PingScan*, a *NodeScan*, and an *Availability* attack using OWL constructs as shown in Fig. 7.7. When specifying the addresses as a datatype property, these inference capabilities could not be leveraged making complex attack detection much more difficult. Both implementations identified the same complex attacks in the example data file used in the trial runs; however, the use of object properties required no additional queries or programming as the complex attacks were all identified using OWL constructs.

Table 7.4: Results of Trial Runs for Address Property.



Task	Time (ms) for Addresses as Datatype Property	Time (ms) for Addresses as Object Property
Create ontology model	1,387.9995	1,507.6309
Read ontology definition files into knowledge base	1,398.9981	1,496.5411
Add address instances to knowledge base	N/A	33.0590
Add packet instances to knowledge base	12,130.2060	6,869.0530
Add alert instances to knowledge base	1,644.6810	1,239.9490
Add stream instances to knowledge base	58,570.1117	68,098.7770
Add PacketCollection instances to knowledge	33,367.9724	38,456.5279



```
<owl:Class rdf:ID="DoSComplex">
<rdfs:comment>
  A complex DoS attack is a Ping scan, Node scan, and
          Availability attack
</rdfs:comment>
<rdfs:subClassOf rdf:resource="#ComplexAttack"/>

<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
              <owl:Class rdf:about="&traffic;NWaddressScanned"/>
      <rdf:Description rdf:about="&traffic;IPaddress"/>
            <owl:Restriction>
              <owl:onProperty rdf:resource="&attack;wasAttacked"/>
              <owl:someValuesFrom rdf:resource="&attack;PingScan"/>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty rdf:resource="&attack;wasAttacked"/>
              <owl:someValuesFrom rdf:resource="&attack;NodeScan"/>
            </owl:Restriction>
                <owl:Restriction>
              <owl:onProperty rdf:resource="&attack;wasAttacked"/>
              <owl:someValuesFrom rdf:resource="&attack;Availability"/>
            </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>

</owl:Class>
```

Figure 7.7: OWL code for the *DoSComplex* class.

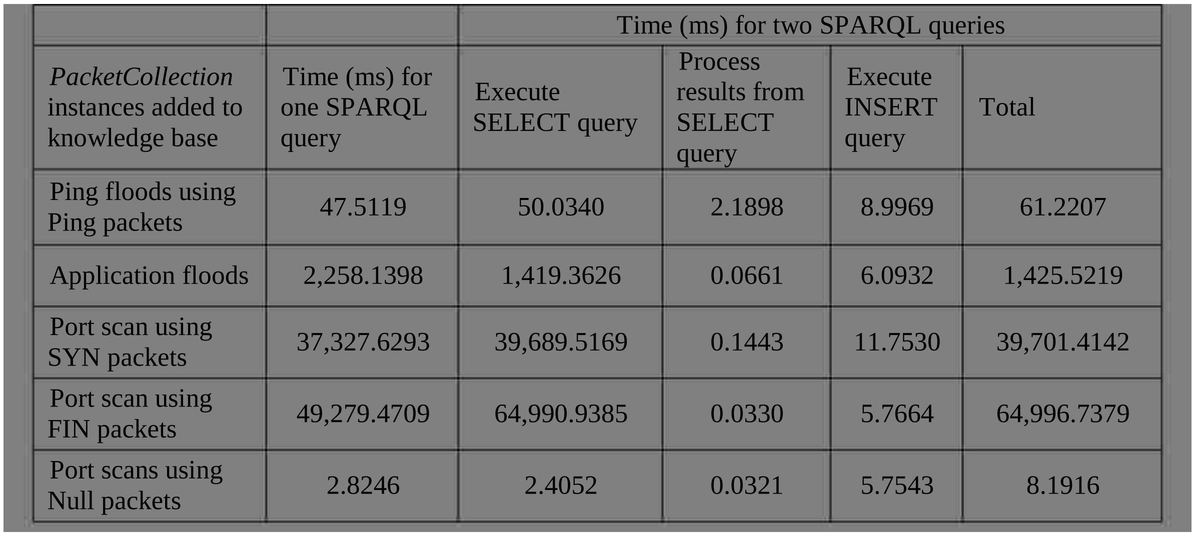
The second implementation decision was compared when inserting instances in the knowledge base for the *PacketCollections* class. These instances represented groupings of similar packets for the detection of simple attack elements, such as flood and scan attacks. The two options implemented and tested were to use multiple SPARQL queries or one SPARQL query. The multiple SPARQL queries option used one SPARQL query to select all the matching instances. The results of this query were then processed programmatically and a SPARQL INSERT statement was constructed and executed. The one SPARQL query option used a single SPARQL query consisting of a combination of the SELECT and INSERT statements. An example of this query is shown in Fig. 7.8.

```
PREFIX traffic: <traffic.owl#>
PREFIX attack: <attack.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT
{
  _:a rdf:type attack:PacketCollection;
        attack:beginDate ?beginDateTime;
        attack:endDate ?endDateTime;
        attack:pcType traffic:PingFloodType;
        attack:hasTargetIP ?destIP;
        attack:pcFrequency ?cnt .
}
WHERE { {
  SELECT  ?destIP   (MIN(?dateTime) as ?beginDateTime)
                (MAX (?dateTime) as ?endDateTime)
                (count(?destIP) as ?cnt)
       WHERE {?pack rdf:type traffic:PingPacket;
                 traffic:dateTime ?dateTime;
                 traffic:hasDestIP ?destIP .
       }
       GROUP BY ?destIP
       HAVING (count(?destIP) > 0)
 }
}
```

Figure 7.8: A SPARQL query to describe a *PingFlood*.

Both implementations were run using a sample data file. Each run resulted in the same instances created in the knowledge base. The results of the trial runs are shown in Table 7.5. For the majority of the types of *PacketCollection* instances added to the knowledge base, the time to execute the single SPARQL query was less than the time to execute two SPARQL queries. The other advantage of the single SPARQL query is the simplicy of the program. The single SPARQL query is a more complex query to write, but it does not require any programming to process the results from the first query and create the INSERT query based on these results, eliminating 115 lines of source code.

Table 7.5: Results of Trial Runs for *PacketCollection* Instances.



		Time (ms) for two SPARQL queries			
PacketCollection instances added to knowledge base	Time (ms) for one SPARQL query	Execute SELECT query	Process results from SELECT query	Execute INSERT query	Total
Ping floods using Ping packets	47.5119	50.0340	2.1898	8.9969	61.2207
Application floods	2,258.1398	1,419.3626	0.0661	6.0932	1,425.5219
Port scan using SYN packets	37,327.6293	39,689.5169	0.1443	11.7530	39,701.4142
Port scan using FIN packets	49,279.4709	64,990.9385	0.0330	5.7664	64,996.7379
Port scans using Null packets	2.8246	2.4052	0.0321	5.7543	8.1916

7.3 Evaluation Methods of the Prototype System

According to Obrst, et al [103], there are many different criteria that can be used to evaluate an ontology. These criteria consist of:

- The ontology’s coverage of a particular domain

- The ontology’s ability to address specific use cases, scenarios, requirements, applications and data sources
- The ontology’s formal properties, such as consistency and completeness
- The ontology’s ability to answer questions, such as “What kinds of reasoning methods can be invoked in the ontology?”

The ontology in the security aspect of this research uses the second criterion: evaluating how well the ontology represents the domain knowledge in specific use cases. This task-based approach is used because it verifies that the ontology represents the domain knowledge concepts and is able to accurately answer queries posed by a domain expert in an application.

The main goal of the validation process is to show that the formal representation can be used to detect complex attacks. The primary method for the evaluation of this criterion is to compare the results of TRIDSO with a current state-of-the-art IDS. Snort was chosen as the system to use for comparison purposes because it is a current state-of-the-art IDS used by many network managers in today’s networks. Another reason that Snort was chosen is because it is the system used by many researchers either as components in their IDS or as a comparative system. The evaluation process used is to run Snort and TRIDSO using the same set of capture files. The attacks detected by each IDS are compared and differences noted.

The Snort configuration used for comparison with TRIDSO is the basic Snort configuration. No special rules were written or installed. Snort generates alerts for many common simple attacks, such as pings and backdoor attempts. The additional configurations added to the Snort installation are the enabling of the TCP/IP checksum mode and the portscan preprocessor. Enabling the TCP /IP checksum mode tells Snort to perform checksum verification for TCP and IP. The portscan preprocessor will generate an alert if a host not in the “home network” (this is a variable that must be configured) initiates more than four port connections within three seconds. This may indicate a possible port scan attack.

Another important criterion in an IDS is its response time. For this reason, the response times, both load and query, for TRIDSO are evaluated. The response times used in the evaluation of TRIDSO are:

1. The time to load the ontology definition files
2. The time to load the raw data instances from the capture file
3. The time to execute a rule
4. The time from the start of TRIDSO until complex attack detection

The last criterion used for evaluating response times is essentially the run-time of TRIDSO against a specific data set (capture file). This is because TRIDSO will load all raw data from the capture file, load other instance via inference and queries, and then identify all complex attacks found in the knowledge base for this data set.

The last evaluation conducted for TRIDSO is its scalability. The amount of network traffic is continually growing. It is important for TRIDSO to run effectively for any size data set. The time to process files of varying size is evaluated.

7.4 Evaluation Results of the Prototype System

7.4.1 Use Case Scenarios

Many different types of attacks were analyzed and tested with Snort and TRIDSO. Snort will identify many of the same simple attacks as TRIDSO, but Snort did not detect any of the complex attacks. Simple attacks make up the steps in a complex attack, so Snort identified some of the steps of the complex attacks, but was never able to generate an alert for a complex attack. TRIDSO was able to detect all of the complex attacks launched in the trial runs.

7.4.1.1 Complex Denial of Service Attack

A Denial of Service (DoS) attack involves an attacker consuming resources on a host or network, thus denying legitimate users access to necessary services. Typically the attacker will identify specific hosts or networks to use as a target of the DoS attack by conducting a ping scan. This combination of a ping scan and DoS attack is categorized as a complex DoS attack. Some complex DoS attacks will also include a step where the attacker will identify a specific port to use in the DoS attack by searching all ports on a found host (a node scan).

The complex DoS attack was simulated with two steps, a ping scan and a simple DoS attack. The ping was performed using nmap [89], a security tool often used to launch attacks. The simple DoS attack was accomplished by sending thirty ping packets to a specific host that was found in the ping scan. The packets for this attack were captured using tcpdump [104].

The capture file is processed by Snort. This produces an alert file containing alerts for all packets that matched Snort rules. For the complex DoS attack, Snort generated eight alerts, two each of the following:

- ICMP PING to the target machine
- ICMP PING NMAP to the target machine
- ICMP Timestamp Requst to the target machine
- ICMP Echo Reply from the target machine

Two ping alerts would not be enough to trigger an alarm to the network manager indicating further analysis is necessary.

TRIDSO has a rule defined in OWL to detect a complex DoS attack. The rule finds all instances of IP addresses that had a ping scan performed against its network and was attacked with an Availability attack (a simple DoS attack). When run with the data from the complex DoS attack conducted, a complex DoS attack against the target host was identified by TRIDSO.

7.4.1.2 The Mitnick Type Attack

A classic complex attack, used by many computer security researchers, is the Mitnick attack. This is a Man-In-The-Middle (MITM) or hijacking attack first performed by Kevin Mitnick. The steps in the Mitnick attack are:

1. Find active hosts to idenfity a target machine (ping scan)
2. Find active ports on the active hosts to identify TCP connections (node scan)
3. Predict the TCP sequence number for the identified TCP connection (TCP connect)

4. Take one host in the TCP connection offline using a DoS attack, typically a Flood attack (DoS)
5. Insert the source machine into the TCP connection by spoofing the host that was taken offline in step 4 (Spoof)

To test a Mitnick attack, an attacker machine, host C, was used to hijack a TCP connection between two other hosts, host A and host B. The specific steps used in this test Mitnick attack are (shown in Fig. 7.9):

1. Ping scan: scan the target network using nmap to identify an active host (host A)
2. Node scan: perform a SYN node scan against an active host (host A), using nmap
3. TCP connect: perform a TCP connect scan against the target host (host A) using nmap
4. Availability: perform a DoS attack against the other host in the TCP connection (host B) to prevent it from responding to host A
5. MITM: perform a MITM attack using ettercap [105] to become the new trusted host to host A in place of host B

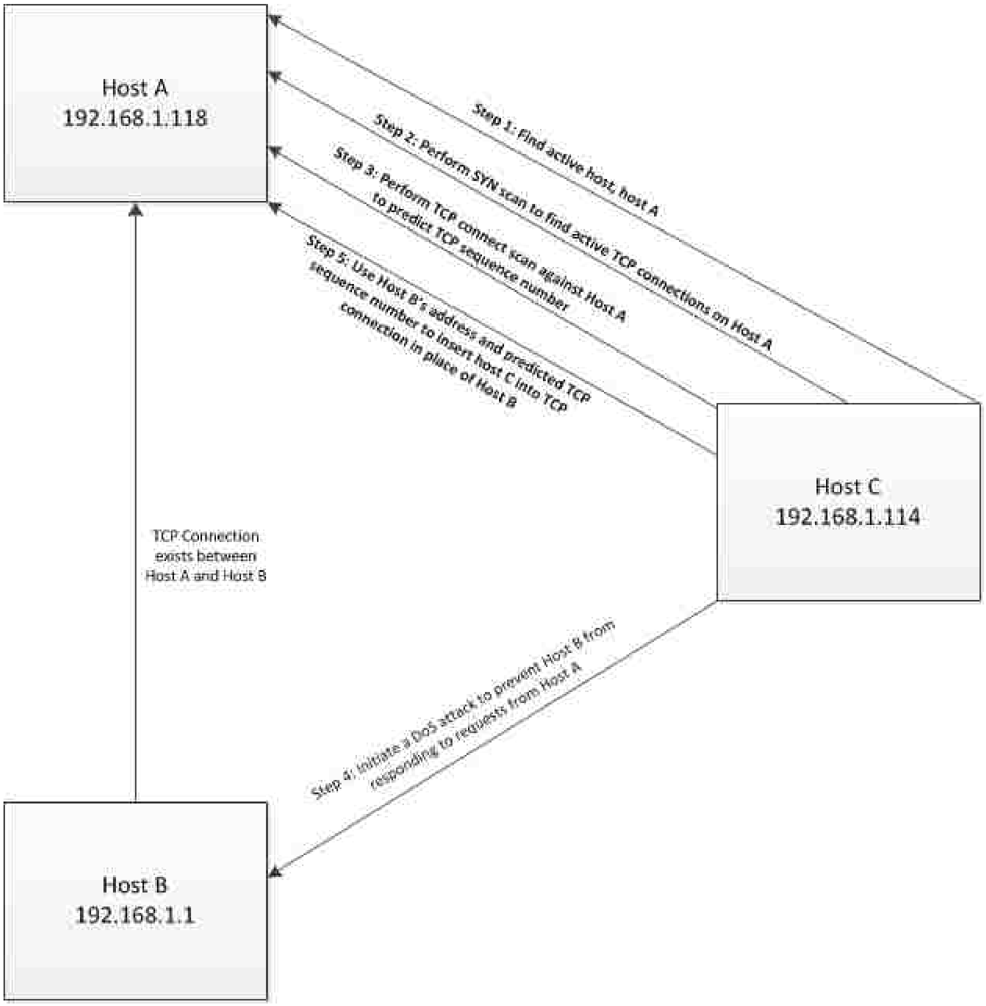

Host A
192.168.1.118
Step 1: Find active host, host A
Step 2: Perform SYN scan to find active TCP connections on Host A
Step 3: Perform TCP connect scan against Host A to predict TCP sequence number
Step 5: Use Host B's address and predicted TCP sequence number to insert host C into TCP connection in place of Host B
Host C
192.168.1.114
TCP Connection exists between Host A and Host B
Step 4: Initiate a DoS attack to prevent Host B from responding to requests from Host A
Host B
192.168.1.1

Figure 7.9: The steps in the test Mitnick attack.

When Snort processed that data capture file for the Mitnick attack, five alerts were generated. The alerts included three unique alerts with two of the alerts repeated for two different hosts, the two hosts in the TCP connection. The alerts generated are:

- TCP Portscan against host A
- ICMP PING against host A and host B
- ICMP PING NMAP against host A and host B

The data was also run through TRIDSO. The OWL rule used to detect a MITM or hijacking attack in TRIDSO finds instances for hosts with all of the following events:

- Host A’s network was scanned
- A node scan attack was performed against host A
- A TCP connect attack was performed against host A
- An availability attack was performed against the other host in the TCP connection, host B
- A spoofing attack was performed against host B

Using this rule, TRIDSO detected the target host of the MITM attack.

In the trial hijacking attack, which was a Mitnick type attack, Snort detected several simple attacks but did not detect any complex attacks. TRIDSO was able to detect a hijacking attack against the target host used in the test Mitnick attack. A query response was generated telling the network manager the IP address of the target host.

7.4.2 Response Time

TRIDSO was run with for many different capture files, simulating a variety of attacks, both simple and complex. The response times for these capture files was analyzed. The response times analyzed are:

- Time to load the ontology definition files into the knowledge base
- Time to load the raw data instances into the knowledge base
- Time to execute a SPARQL query to insert additional instances into the knowledge base from existing instances
- Time to execute a query against the knowledge base to retrieve instances

- Time from the start of TRIDSO until complex attacks are detected (run time)

For the query times, sample queries were used for analysis.

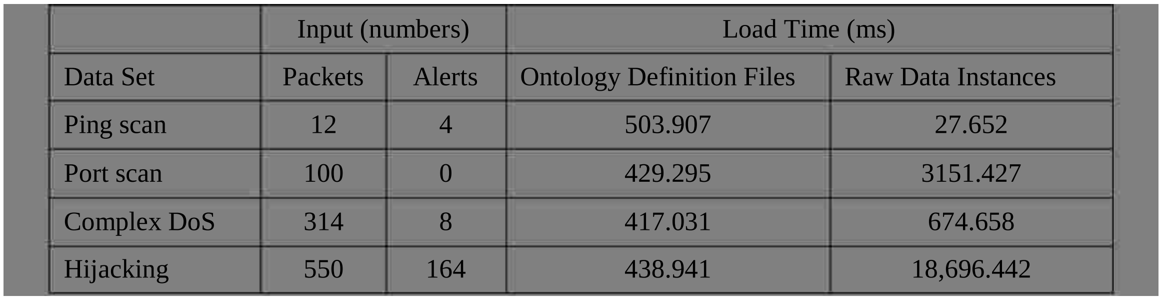
Several of the capture files were selected for evaluation purposes. These include a sampling of both simple and complex attacks. The capture files selected included the following types of attacks:

- A ping scan
- A port scan
- A complex Denial of Service (DoS) attack
- A complex hijacking attack

The load times for these data sets are shown in Table 7.6. The durations to load the ontology definition files are reasonably constant. This is to be expected since the ontology definition files are static for all runs of the system.

The time performance to load the raw data instances into the knowledge base varied for each data set. Typically, as the number of raw instances (packets and alerts) increases, the time to load the raw instances also increases. There will be some fluctuation in this load time due to the variation in packet types in the raw data and the association class definitions.

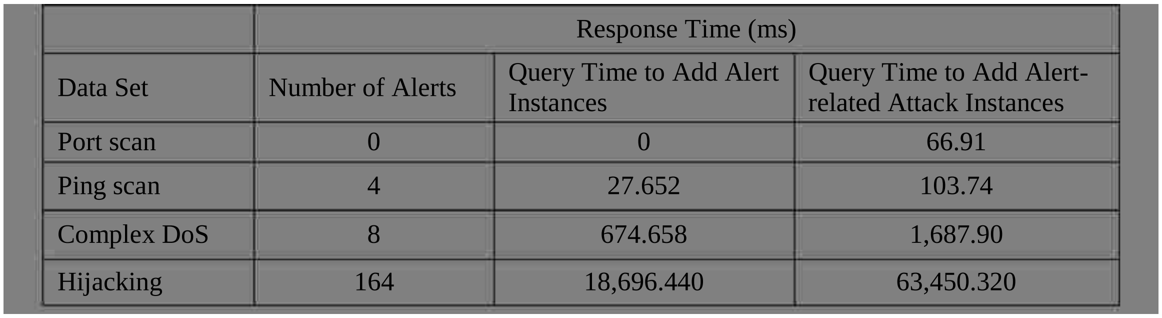
Table 7.6: Load Time Performance for Trial Data Sets.



	Input (numbers)		Load Time (ms)	
Data Set	Packets	Alerts	Ontology Definition Files	Raw Data Instances
Ping scan	12	4	503.907	27.652
Port scan	100	0	429.295	3151.427
Complex DoS	314	8	417.031	674.658
Hijacking	550	164	438.941	18,696.442

Table 7.7 shows the response time performance for alert instances for the sample data files. These results clearly show that the time to add instances to the knowledge base is directly related to the number of alerts in the raw data set. As the number of alerts increases, the time to add the alert instances to the database increases.

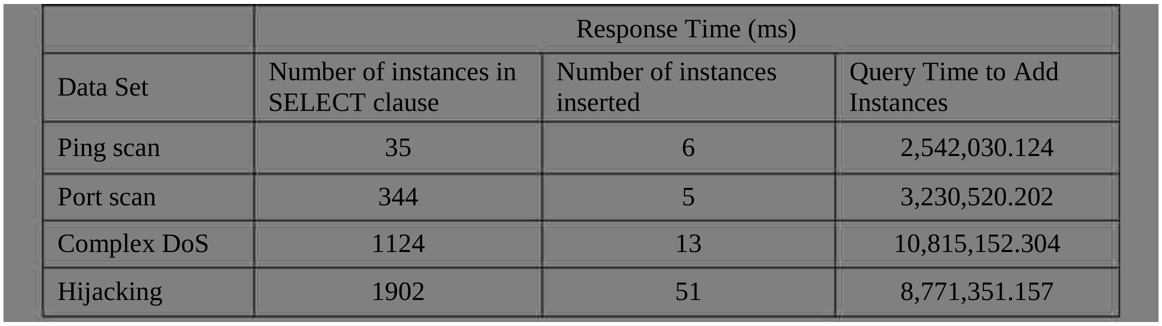
Table 7.7: Alert Query Response Time Performance for Trial Data Sets.



	Response Time (ms)		
Data Set	Number of Alerts	Query Time to Add Alert Instances	Query Time to Add Alert-related Attack Instances
Port scan	0	0	66.91
Ping scan	4	27.652	103.74
Complex DoS	8	674.658	1,687.90
Hijacking	164	18,696.440	63,450.320

The response time data for the query to add *PacketCollection* instances to the knowledge base for the sample data files are shown in Table 7.8. Adding these instances involves selecting instances from a variety of classes based on the instances matching specified criteria and then inserting the appropriate instance to the *PacketCollection* class. The time to add the *PacketCollection* instances increases as the number of instances involved in the query increases.

Table 7.8: Query Response Time Performance for Trial Data Sets.



	Response Time (ms)		
Data Set	Number of instances in SELECT clause	Number of instances inserted	Query Time to Add Instances
Ping scan	35	6	2,542,030.124
Port scan	344	5	3,230,520.202
Complex DoS	1124	13	10,815,152.304
Hijacking	1902	51	8,771,351.157

The total detection time, which is the time from the start of TRIDSO to the time it takes to detect all complex attacks in the data set, is shown in Fig. 7.10. As the size of knowledge base, which is the number of raw and total instances, increases, the time it takes to detect complex attacks also increases. The response time data indicates that TRIDSO is not able to detect complex attacks in real-time, which is discussed in the next section and future work.

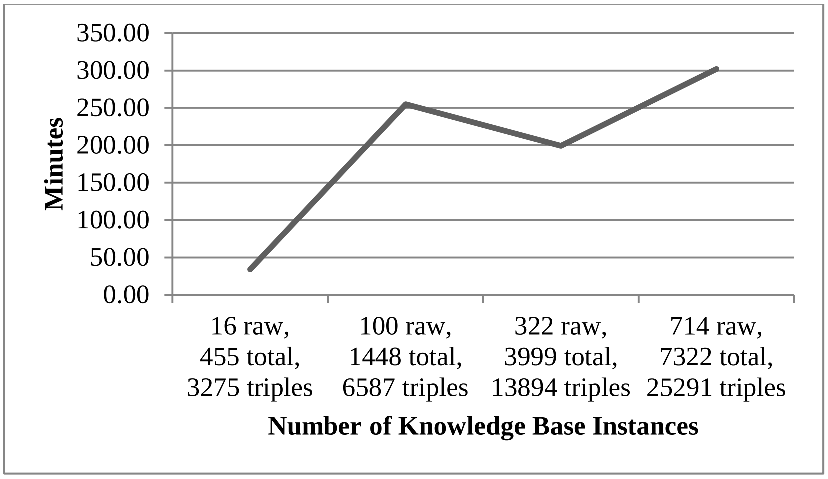


Figure 7.10: The time performance of complex attack detection by TRIDSO.

7.4.3 Scalability

The total run time data of TRIDSO against a variety of data file inputs are shown in Fig. 7.11. This evaluation highlights one limitation with TRIDSO; the scalability of the system. TRIDSO utilizes Jena, which is not a scalable environment; it is acceptable for use in the proof-of-concept system but a full implementation of TRIDSO would require a different environment.

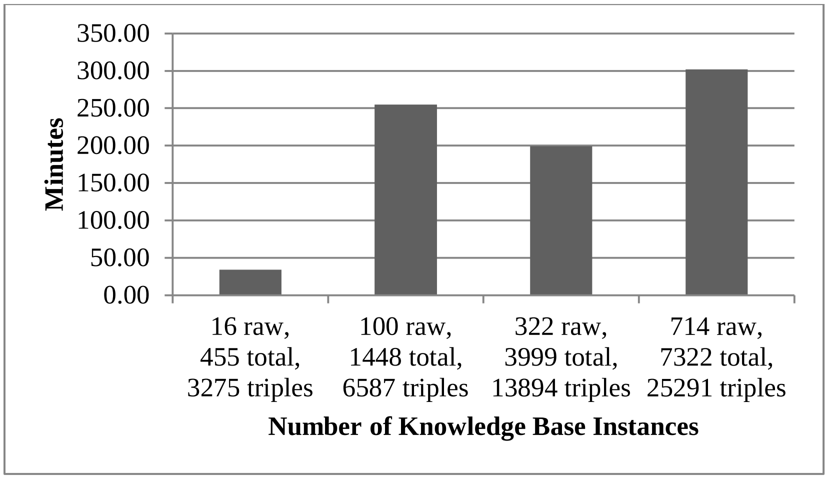


Figure 7.11: The run time performance of TRIDSO.

The scalability issue in TRIDSO prevents it from detecting attacks in real-time; however, it can still be beneficial. Detecting attacks, even if it is post-occurrence, is beneficial to the security of a network because there can be many lessons learned post-attack. The most beneficial lesson learned is the current vulnerabilities of the network and its nodes. When TRIDSO detects an attack, it informs the network manager how attackers are attempting to attack the network. Even if an attack is successful and undetected in real-time, the network manager will now know how the attack occurred so future occurrences can be prevented.

Snort also experiences scalability issues. The number of rules in Snort has been growing exponentially in the last few years, according to statistics gathered (see Fig. 7.12). This leads to more complexity in the management of Snort. The large rule set also leads to a larger run time. The way Snort continues to detect in real-time with the large rule set is to skip packets when the Snort processor cannot keep pace with the incoming packets. This will lead to a higher rate of false negatives, which means that attacks may go undetected by Snort.

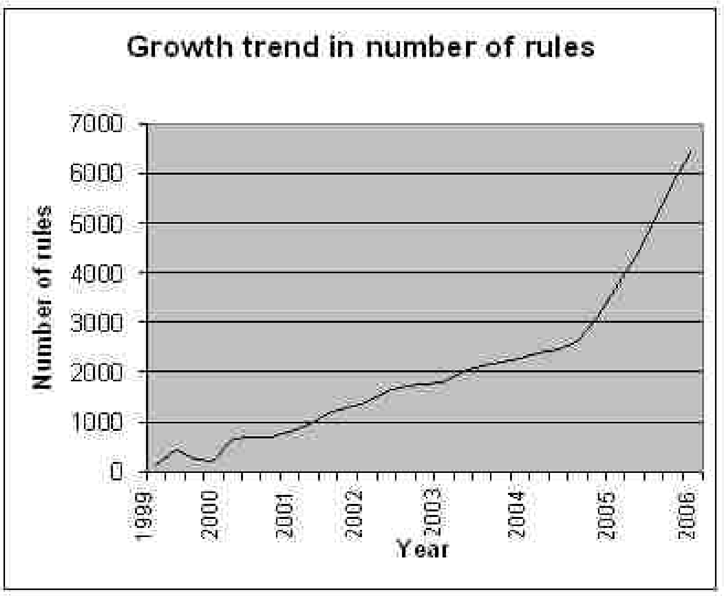
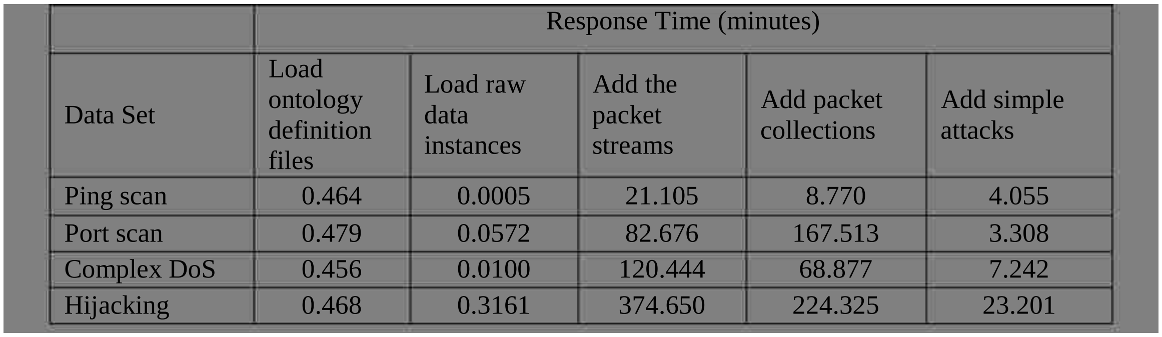


Figure 7.12: The growth trend of the number of rules in Snort [51].

It is important to analyze the time performance for each aspect of the prototype system to determine what aspect of the system is contributing most to the large response delays. The tasks in TRIDSO contributing the most time toward the overall run time are adding the instances for traffic streams and packet collections. Table 7.9 shows the individual response delays for the primary aspects of the system, including all the aspects contributing the most to the total run time. The initialization of the knowledge base includes the loading of the ontology definition files and the raw data instances. These aspects of the system are processed programmatically using the Jena API. The remaining response delays, which contribute the most time to the overall run time, are for the aspects of the system that include the SPARQL queries. These delays represent the addition of many of the simple attack instances to the database, including the packet streams, packet collections, and miscellaneous simple attacks.

Table 7.9: Time Performance for Trial Data Sets.



	Response Time (minutes)				
Data Set	Load ontology definition files	Load raw data instances	Add the packet streams	Add packet collections	Add simple attacks
Ping scan	0.464	0.0005	21.105	8.770	4.055
Port scan	0.479	0.0572	82.676	167.513	3.308
Complex DoS	0.456	0.0100	120.444	68.877	7.242
Hijacking	0.468	0.3161	374.650	224.325	23.201

As seen from Table 7.9, the majority of the overall run time is contributed to the execution of the SPARQL queries, specifically the queries to add the packet streams and packet collections. For each of these tasks, there are multiple complex queries with each query often involving multiple select statements nested within an insert statement. For instance, there are five SPARQL queries to add the packet streams, sixteen to add the packet collections, and four to add the miscellaneous simple attacks. The time to create the instances for complex attacks is not shown in the table because these instances are created using OWL code and incur very minimal overhead.

The system developed, TRIDSO, is a prototype system to test the feasibility of utilizing the developed formal representation in detecting complex attacks. TRIDSO was developed as a quick-and-dirty prototype system; no optimization techniques were included in the system. Optimizing aspects of the system will decrease the run-time and help with the scalability problem. For instance, one major contributor to the run-time is the addition of the instances to the Streams classes. This is done using numerous

SPARQL queries. Many of these queries can be run in parallel on multiple processors. Adding multi-processing functionality, specifically to the execution of the SPARQL queries, will decrease the overall run-time and alleviate some of the scalability problem.

Another optimization that can be added to TRIDSO is the rule engine used for the model. The ontology API in Jena is used, which uses an ontology model. The specific ontology model used by TRIDSO is the OWL_MEM_RULE_INF model. This specific ontology model supports the OWL Full language and stores the model in memory. The reasoner used by this model is a rule-based reasoner with OWL rules. By using a reasoner in Jena, the triples that are asserted by the inference algorithm are added to the model, thus becoming part of the knowledge base. This will increase the overall run-time of TRIDSO because there will be more triples in the knowledge base, requiring additional processing by any query.

The reasoner used by the model in TRIDSO is an OWL rule reasoner. This reasoner is not well suited for large ontologies, but it supports the contructs available in OWL Full, such as the Boolean constructions (unionOf, intersectionOf), someValuesFrom and the cardinality restrictions. It was selected for use in the prototype system to allow experimentation with various OWL Full constructs during the development of the formal representation. The use of this reasone leads to poor performance: “the rules implementing the OWL constructs can interact in complex ways leading to serious performance overheads for complex ontologies” [106]. To utilize TRIDSO in a production environment a different reasoner needs to be selected and implemented in TRIDSO, which would lead to an improvement in the performance.

The OWL rule reasoner used in TRIDSO supports a hybrid approach, using both forward and backward chaining, to support inference. Forward chaining is used on the raw instances in the knowledge base to infer additional triples. Backward chaining is invoked to answer queries, which may be invoked by backward rules or when the forward chaining engine asserts new backward rules. Utilization of a different reasoner may lead to performance improvements. Analysis must be completed to determine the appropriate reasoner for an implementation of the formal representation in a production system. Additional options for addressing the scalability problem are discussed in the section 8.2.2 (Future Works for Representation of Complex Attacks).

7.5 Chapter Summary

A prototype system, TRIDSO, was developed to test the formal representation designed for detecting complex attacks. TRIDSO used a set of SPARQL rules to incorporate additional functionality not achievable with OWL. These rules were used to add instances to the knowledge base based on existing instances, such as packet collections (scans, floods, etc.) and attacks based on the Snort-generated alerts.

TRIDSO was run with a variety of sample attacks, both simple and complex. The output was compared with Snort, a state-of-the-art IDS used by many security administrators and researchers. While Snort was able to detect some of the simple attacks, TRIDSO was able to detect more simple attacks. TRIDSO was also able to detect all the complex attacks in the data sets, while Snort was not able to detect any of the complex attacks.

Response delays of the sample runs in TRIDSO were analyzed. As the number of knowledge base instances and the data set size increases, the response delays increase. These delays underline the fact that the current implementation of TRIDSO is not scalable to a real-time detection environment. Additional research and further development of TRIDSO is required to determine if TRIDSO could be adapted for real-time intrusion detections. Even if that is not possible, the use of TRIDSO is a valuable asset to a network manager. It allows the network manager to understand weaknesses in the network and take corrective action to prevent future attacks.

Chapter 8 **Conclusions and Future Work**

Networks and the services they provide have become a ubiquitous part of computing for virtually any computer users. Users have the expectation that the network will be available around the clock. As shown in previous discussions, network management today is a significant challenge. This challenge includes availability, network management, and security.

One challenge facing network management is the large variety of components on networks. These components may be on different tiers of the network (wired, ad hoc, WSN) and from different manufacturers (Cisco, Nortel, etc.). This characteristic of Heterogeneous Multi-tier Networks makes network management an arduous task.

Any network threat represents the possibility that network availability becomes compromised. Identifying attacks against the network is a challenge facing network managers. The users' expectations of the always-available network and the organization's expectation of securing its data make security a high-priority task. If a device is attacked, it may become unavailable to the users or have data compromised. When the victim device is a network device, the consequences are compounded as this affects many devices on the network and possibly the entire network.

As attacks become more common and complex, detecting attacks becomes more difficult. A complex attack consists of a sequence of simple attacks. Current Intrusion Detection Systems (IDSs) often detect simple attacks, comprising some of the steps in a complex attack, but do not detect complete complex attacks. The development of an IDS capable of detecting complex attacks can improve the network manager’s ability to ensure an available, secure network.

We have explored the question of whether an ontological approach to network management is effective. Using an ontological approach enables us to create a single NMS for all deployed devices. We have shown both the appropriateness and feasibility of using ontology as the basis for a NMS.

Another important aspect of network management is the management of security. Using the ontological representation, it is possible to detect more simple and complex attacks. Detecting more attacks will ensure better availability and security of the organization’s network and data. Even though the developed IDS detects attacks after they occur, it is still an important tool in network security. The information learned from a detected attack, even post-occurrence, will help with future iterations of network security.

8.1 Conclusions and Contributions

A framework based on ontological representations was designed to manage and provide interoperability among components of Heterogeneous Multi-tier Networks. Four contributions are linked to the network configuration and security management: (1) adaptable knowledge base, (2) analysis of performance, (3) ontological representation for complex attacks, and (4) evolution of ontological representation with extensible heuristics.

The adaptable knowledge base of the first contribution significantly enhanced the ability to manage Heterogeneous Multi-tier Networks as they evolved in number and complexity. An ontology based Network Management System (NMS) addressed potential challenges as new technologies and dynamic components were introduced to heterogeneous network managers. The reporting process provided seamless integration of support to manage Heterogeneous Multi-tier Networks from the daily management perspective as additional data was collected. We created a prototype of the ontology-based NMS to prove its viability in forms of effectiveness and performance. The prototype demonstrated the network management as an n:1 improvement in the toolset required for management of a HMN, where n is the number of different device types, or tiers, within the network. By current industry standards, NMSs provide management capability at the device-type level whereas the new HMNMS in this research provided systemic management of the entire HMN. This new HMNMS will allow a network manager to obtain a systematic view of the HMN instead of having to manage the individual component networks.

Our second line of investigation asks the question of whether a HMNMS would degrade network performance. We developed a model to evaluate performance based on a theoretical queuing framework. This analysis of a HMN was conducted to verify the model type and identify bottlenecks. The analytical model in this scenario was then utilized to prove that the bottleneck in a Heterogeneous Two-tier Network (wired and ad hoc tiers) was the ad hoc gateway and not the Heterogeneous Multi-tier Network Management System (HMNMS). The model also demonstrated that the HMNMS did not have an adverse effect on the HMN. Network designers may utilize this analytical model to determine the bottleneck in a HMN as well as the number of gateway devices required while maintaining optimal performance. Thus our second contribution is based on the discovery that a HMNMS will not degrade performance of a HMN.

The basis for the ontological representation of attacks was based on generalized attack trees developed by this researcher. The generalized attack trees for complex attacks were defined based on researching specific attacks and recognizing attack patterns. The ontological representation provided more flexibility because its declarative representation allowed for augmentation without impacting other aspects of the system. This in turn allows the ontology to be extended by others doing related research therefore extending the knowledge and enabling the detection of evolving attack strategies. Traffic data was used to develop and utilize the formal representation allowing for complex attacks and attack attempts to be detected, which provided flexibility over a programmatic approach.

As attack trees were developed, heuristics were established that effectively implement the attack recognition process. These heuristics represent a fourth contribution as an attempt to codify meta-characteristics of attacks. By using ontology constructs available in OWL (the formal representation) and a query language (SPARQL), manipulation of the ontology became easily modifiable and extendable with the addition of rules to detect additional complex attacks. A prototype system (TRIDSO: Traffic-based Reasoning Intrusion Detection System using Ontology) was developed to show the feasibility of using the developed formal representation with a set of heuristics to detect complex attacks and attack attempts. In the analysis of data, results showed the prototype system detected more simple and complex attacks and attack attempts than a current state-of-the-art system that was used for comparison.

The combination of the formal representation and corresponding set of heuristics developed identify more simple and complex attacks than a current state-of-the-art IDS. This allows a network manager to respond to the simple and complex attacks detected. The manager can add additional security measures in a future iteration of security measures for the network allowing for the prevention of more simple and complex attacks.

8.2 Future Work

This research has led to several significant contributions in network management and like most research has also led to several open questions. There are many different areas for future work in both areas of research, management of HMNs and IDSs. An integral part of the future work is to merge the HMNMS and TRIDSO into one NMS for HMN. This NMS will begin to provide configuration and security management for HMNs. It can eventually provide all management areas by adding performance, fault, and accounting management.

8.2.1 A Heterogeneous Multi-tier Network Management System

The developed HMNMS is a basic conceptual prototype to show the feasibility of using ontology in a NMS for HMNs. The defined ontologies will continue to be refined, maintained and extended. Additional properties can be included for the four device types included in the initial ontologies. Some of the expanded properties will be the interfaces and connections in the network so that a logical network map can be drawn from the discovered topology.

8.2.1.1 Add Additional Tiers

Additional device types can also be added to the NMS, such as additional wired manufacturers or sensors for WSNs. The WSN device type is included in the developed ontology; however, it was never implemented in the prototype system or tested. In the simulation tests, WSN device type instances were added to the knowledge base statically using OWL code. Future work will be to test the WSN portion of the ontology with a live WSN. This portion of the network will then be added to the analytical model so a performance analysis can be conducted for a three-tier HMN.

8.2.1.2 Extend the System to Other Network Management Areas

The prototype NMS focused on topology management but the HMNMS can be extended to include additional management tasks, such as more configuration management tasks or performance management. Areas that would be most beneficial to network managers are fault and security management. Extending the NMS to security management has been tested in the other part of this research by developing a formal representation for complex attacks and implementing the representation using ontology. There is some discussion about merging the two ontology systems (the HMNMS and TRIDSO) later in this chapter.

8.2.1.3 Automatically Convert MIBs to OWL

One way to enhance the HMNMS is to automate some aspects of the system development, particularly some of the ontology definitions. Future work will include incorporating the creation of the OWL files for the device types into the HMNMS. The SNMP MIBs can be converted to OWL and used in the HMNMS.

It is possible to automatically create OWL ontology files from XML data. Bohring and Aver [107] proposed a framework to translate XML data to OWL. Their work converts the tree structure in XML to the corresponding class hierarchy in OWL. The difficulty is the fact that OWL is more expressive than XML, making some of the mapping difficult. It is necessary to determine the appropriate representation in OWL for a less-expressive representation in XML. There is a desire to leverage the expressive nature of OWL and convert some structures in XML that do not have a direct mapping to OWL. For example, Bohring and Aver assume that there are some relational structures in XML, such as nested tags. These mappings are not as straight-forward and require some assumptions and/or experimentations. In this instance, Bohring and Aver mapped a nested tag in XML to an ObjectProperty in OWL.

Another project that has developed an automatic conversion process, which may be used in the HMNMS, is the AstroGrid-D project from the German Astronomy Community Grid (GACG) [108]. The AstroGrid-D project required the data to be converted to RDF prior to being uploaded to the astronomy application. Two different options were defined to do this transformation and both will be evaluated for use in converting data for use in the HMNMS. The first one is an XSL stylesheet (xml2rdf.xsl) that will convert XML files to RDF files. XSL (Extensible Stylesheet Language) [109] is a series of recommendations for transforming XML. The second option uses a Java package OwlMap. This package consists of two programs. One program, XS2DAMLOIL converts XML to OWL format and the other one, XML2RDF, converts XML to RDF.

Preliminary investigation was conducted on this research but additional work is required. Some of the MIBs are currently available in the XML format. For MIBs not available in XML format, they can be converted to XML or XML Schema (XSD) using smidump, which is a program available as part of the libsmi library. The libsmi library is a library that provides access to SMI MIB information through various functions. After obtaining an XML version of the MIB, it can be converted to RDF, which can be used as the OWL definition files in the HMNMS. This was tested with a few MIBs using the XS2DAMLOIL program. Preliminary results proved that this was possible but more research is required to see if the results are practical for use in the HMNMS.

8.2.1.4 Utilization of the Analytical Model

Another area of future work is the utilization of the developed analytical model. The analytical model is used to find the network capacity. In this research it was used to evaluate the performance of a heterogeneous two-tier network. Specifically, the analytical model was used to identify the bottleneck in a heterogeneous two-tier network.

Future work will employ the analytical model for other performance evaluations. One possible use that may be developed is to evaluate the performance and identify bottlenecks when additional tiers are added to the network. For instance, a Wireless Sensor Network (WSN) may be added as an additional tier. This will require a WSN gateway, which may introduce a potential bottleneck. Another performance metric that may be evaluated using the analytical model is the determination of the number of nodes each gateway, ad hoc or WSN, can efficiently support. This evaluation may be used by the network manager in determining when another gateway must be added for continual, efficient performance of the gateway node(s).

The analysis performed used constant parameters, such as the packet size and the number of response packets generated. It was also assumed that there was no packet loss.

Future work will include conducting dynamic end-to-end network performance by varying parameter values. Some of the parameters that may vary in future evaluations are the number of management packets sent to different nodes or node types, packet size, and the number of response packets generated for each management request. Future work will also introduce some probability of packet delays and packet loss.

The analytical model can also be used for traffic modeling. This requires using a large amount of complex traffic in the model. The performance evaluations conducted in this research used a constant traffic rate. For traffic modeling, the traffic should be varied in type, packet size and rate. Traffic modeling can use the model to determine buffer occupancy statistics, queue wait times, and blocking probabilities.

8.2.2 Representation of Complex Attacks

This research designed and developed a formal representation for complex attacks, which can easily be extended due to the use of ontology. A prototype system was developed to demonstrate the viability of using the formal representation in an IDS. This prototype system is in its infancy and may continue to be developed.

8.2.2.1 Incorporation of Additional Subsystems

One clear extension for TRIDSO is to incorporate the remaining subsystems, the device subsystem and the vulnerability subsystem. An initial version of the device subsystem was developed as part of this research and utilized in the HMNMS discussed in chapter 3. The ontology utilized in the HMNMS forms the foundation for the device subsystem in TRIDSO. This ontology will be extended and incorporated into TRIDSO.

The vulnerability subsystem requires development. It will utilize existing repositories of known vulnerabilities, such as NIST’s NVD (National Vulnerability Database). An optimal solution is to utilize existing research that creates ontology instances from these repositories. One such work is OVM (Ontology for Vulnerability Management) [100].

8.2.2.2 Determining Threshold Values

There are several threshold values utilized in TRIDSO and the determination of the optimal value to use for each is left for future investigation. Specifically, there are four threshold values used in TRIDSO:

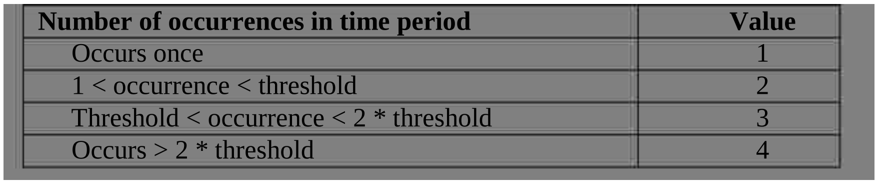
1. Rate category – determining the appropriate value for the rate category in the coloring scheme [91]
2. Flood attack occurrences – the number of occurrences of a specific packet type before it is identified as a flood attack
3. Scan attack occurrences – the number of occurrences of a specific packet type before it is identified as a scan attack
4. Timeframe – the length of time to use for including packets when identifying attacks

Initially, optimal values will be determined for each of these threshold values and remain static.

The next step will be to incorporate a *training phase* into TRIDSO making it self-learning. The system starts using the identified threshold values. As the system runs, the threshold values are adjusted based on observed traffic conditions. For example, consider a ping flood attack. Let’s assume the threshold value identified for a flood attack is twenty-five occurrences in the specified timeframe. For a corporate network, where ping is only used by network and system administrators, this value may be too high. For a university network, where computer science courses may use ping as a teaching tool, this value may be too low. As the system runs, an algorithm would be utilized that looks at historical traffic data and adjusts the threshold values accordingly.

A coloring scheme [91] was developed that will be incorporated into TRIDSO. The first threshold value is used in the coloring scheme to determine the value assigned to the rate category. The threshold is used to determine the number of occurrences of a specific packet type, as shown in Table 8.1. The appropriate threshold should be determined prior to incorporating the coloring scheme into TRIDSO.

Table 8.1: The Coloring Scheme’s Rate Category Values.



Number of occurrences in time period	**Value**
Occurs once	1
1 < occurrence < threshold	2
Threshold < occurrence < 2 * threshold	3
Occurs > 2 * threshold	4

The second and third threshold values are similar. They both deal with the number of occurrences of a specific packet type to identify flood and scan attacks. A few ping packets are often not an issue as ping is a common troubleshooting tool; however, ping is also a common tool for attackers. It is important to determine the best value for this threshold. A threshold that is too high may lead to false negatives, indicating an attack occurred but was not identified. A threshold that is too low leads to true positives, indicating an attack was identified but it was not an attack. These situations can lead to additional analysis time by the network manager and possibly unnecessary network down time.

The fourth threshold value to be determined is the length of time to use when identifying attacks. For example, when looking for a ping flood attack, if the flood threshold is twenty-five, the system looks for the occurrence of twenty-five ping packets to the same host or network. The timeframe determines when these packets occur. Do they occur within five milliseconds of each other? Five minutes? Five hours? This is a critical question because twenty-five ping packets to the same host over five hours is usually not a problem; however, twenty-five in five milliseconds may indicate a possible denial of service attack against the host.

When the optimal value for this timeframe threshold is determined, it will be used in detecting possible attacks. As the research is conducted in identifying this optimal value, it may be determined that several timeframe thresholds are necessary. A threshold value of ping packets to the same host or network in five minutes is probably not enough to indicate a possible ping flood attack; however, a threshold value of ping packets to different hosts on the same network in five minutes may indicate a possible ping scan attack. It may be necessary to use different timeframe thresholds for different types of attacks.

Another use of the timeframe threshold in TRIDSO is determining the occurrence of a complex attack. In this case, the time from the first node being in an attack tree to the time the root node is colored will be measured. Finding the most effective timeframe is a critical step in complex attack identification.

8.2.2.3 Probabilistic Complex Attack Detection

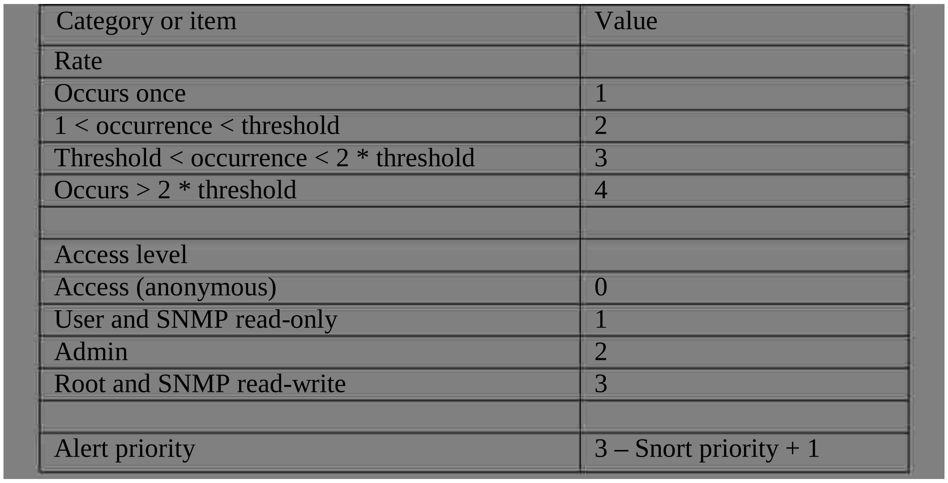
A coloring scheme [91] will be incorporated into TRIDSO. This coloring scheme will allow for the incorporation of probability in the detection process.

When a node in an attack tree is identified as having occurred, the node is colored. A three-color scheme is used: 1) green indicates no attack occurred, 2) yellow indicates an attack may have occurred, and 3) red indicates that an attack most likely occurred.

All nodes are assigned a color based on a priority assigned to the attack element for that node. The priority is determined based on three categories of analysis. These categories are shown in Table 8.2, with their corresponding values, and are explained below.

The first category is the rate, which indicates how often the element occurred in a time period. The rate is assigned a value of one through four based on a threshold value. A value of one is assigned if the attack element occurred once in the time frame, two if it occurred more than once but less than the threshold, three if it occurred more than the threshold but less than twice the threshold, and four if it occurred more than twice the threshold. The most effective threshold value has not yet been determined.

Table 8.2: Attack Element Priority [91].



Category or item	Value
Rate	
Occurs once	1
1 < occurrence < threshold	2
Threshold < occurrence < 2 * threshold	3
Occurs > 2 * threshold	4
Access level	
Access (anonymous)	0
User and SNMP read-only	1
Admin	2
Root and SNMP read-write	3
Alert priority	3 – Snort priority + 1

The access level category is the user (for a host) or privilege (for a network device) level gained, or possibly gained, by the attack. Four different access levels are utilized. The first level is similar to anonymous and gives a remote user access to the device or resource, such as a web user on a web server. This first level is assigned a value of zero. Next is the user level, with a value of one, which is a typical user on a system. The admin level has a value of two and the root level has a value of three. These two levels have been separated; even though they are synonymous on many systems, some systems separate the two. For example, the Windows operating system admin user, while often considered the same as root on the UNIX operating system, is different because some operations on Windows require local administrator access. The other type of access is that provided by SNMP. Read-only access provided by SNMP is equivalent to the user level and read-write access is equivalent to the admin level.

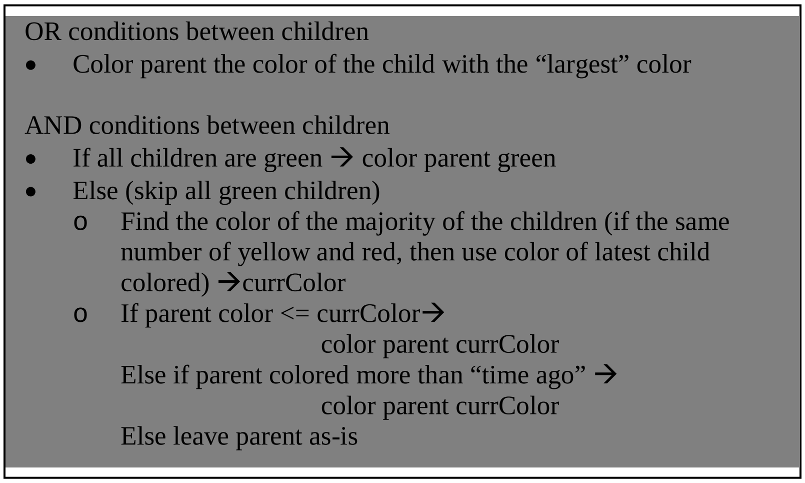
The last category considered in the coloring scheme is the alert priority. This is based on the priority assigned to the alert produced by Snort, if one is assigned. The priority for an alert in Snort can have a value of zero through three, with zero being the highest priority. The coloring scheme assigns zero the lowest priority, so the following equation is used to convert the alert priority to the appropriate value in the coloring scheme:

$$\text{value} = 3 - \text{Snort\_priority} + 1 \tag{8.1}$$

The reason to add one is because there is a need to not have a value of zero assigned to the alert category since Snort assigned it a priority value, thus considering it of some importance.

The priority of the attack element is calculated by adding the values of the three categories. The node in the attack tree(s) corresponding to the attack element is colored appropriately. To determine the appropriate color, the possible values, zero through eleven, which is the total possible value for the attack element priority, are divided evenly. A priority less than or equal to three causes the node to be colored green, a value from three to eight colors the node yellow, and a value of eight or more colors the node red.

After all the affected nodes are appropriately colored, the coloring propagates up the attack tree. The parent nodes are colored based on the colors of the children nodes. The coloring algorithm (shown in Fig. 8.1) is based on empirical observations of results from test iterations of a simulation program developed to design the proposed coloring scheme. If the children nodes have an OR condition in the attack tree, then the parent node is colored with the “largest” color, with a descending order of red, yellow, green. If the children nodes have an AND condition in the attack tree, propagating the color to the parent becomes more complex. If all the children are green, then the parent is colored green; otherwise, the green nodes are excluded in the determination of the parent color.



OR conditions between children
- Color parent the color of the child with the “largest” color

AND conditions between children
- If all children are green → color parent green
- Else (skip all green children)
    - Find the color of the majority of the children (if the same number of yellow and red, then use color of latest child colored) →currColor
    - If parent color <= currColor→
      color parent currColor
      Else if parent colored more than “time ago” →
      color parent currColor
      Else leave parent as-is

Figure 8.1: The coloring scheme algorithm [91].

To begin color analysis for the parent node, the majority color of the children nodes is determined. If there are an equal number of yellow and red children, then the color of the node that was just colored is used as the majority. If the parent node is the same color as the majority color or it was colored less than a time threshold ago, then the parent node remains the same color. If the parent color is less than the majority color, then the parent is colored that color. The most effective value of the time threshold used has not yet been determined.

As an example (see Fig. 8.2), consider a situation where an attack occurs that scans all hosts on the network. There is also a telnet to the SNMP port of a SNMP-managed node. The algorithm determined these nodes should be colored yellow. There was also a port scan that occurred with high occurrence in a time period, so that node was colored red. The colors were then propagated to the parent nodes, resulting in the colored attack tree. Uncolored nodes in the attack tree indicate the attack was not detected.

Probability will be incorporated into the rate category. This category is assigned a value (1-4) based on how often the element occurs in a time period in relation to a threshold value. The assignments are shown in Table 8.2.

Based on the rate value, a probability will be assigned to the occurrence of the simple attack. The color assigned will be associated with a probability. The most appropriate probability to assign each value and color is also part of future work, but a baseline is used for discussion. This baseline associates a probability to the various values of the rate category according to Table 8.3. These probabilities then correspond to the appropriate color.

Table 8.3: Probabilities for Rate Category Values.

Number of occurrences in time period	Value	Probability	Color
Occurs once	1	0 – 24	Green
1 < occurrence < threshold	2	25 – 49	Yellow
Threshold < occurrence < 2 * threshold	3	50 – 74	Yellow
Occurs > 2 * threshold	4	75 - 100	Red

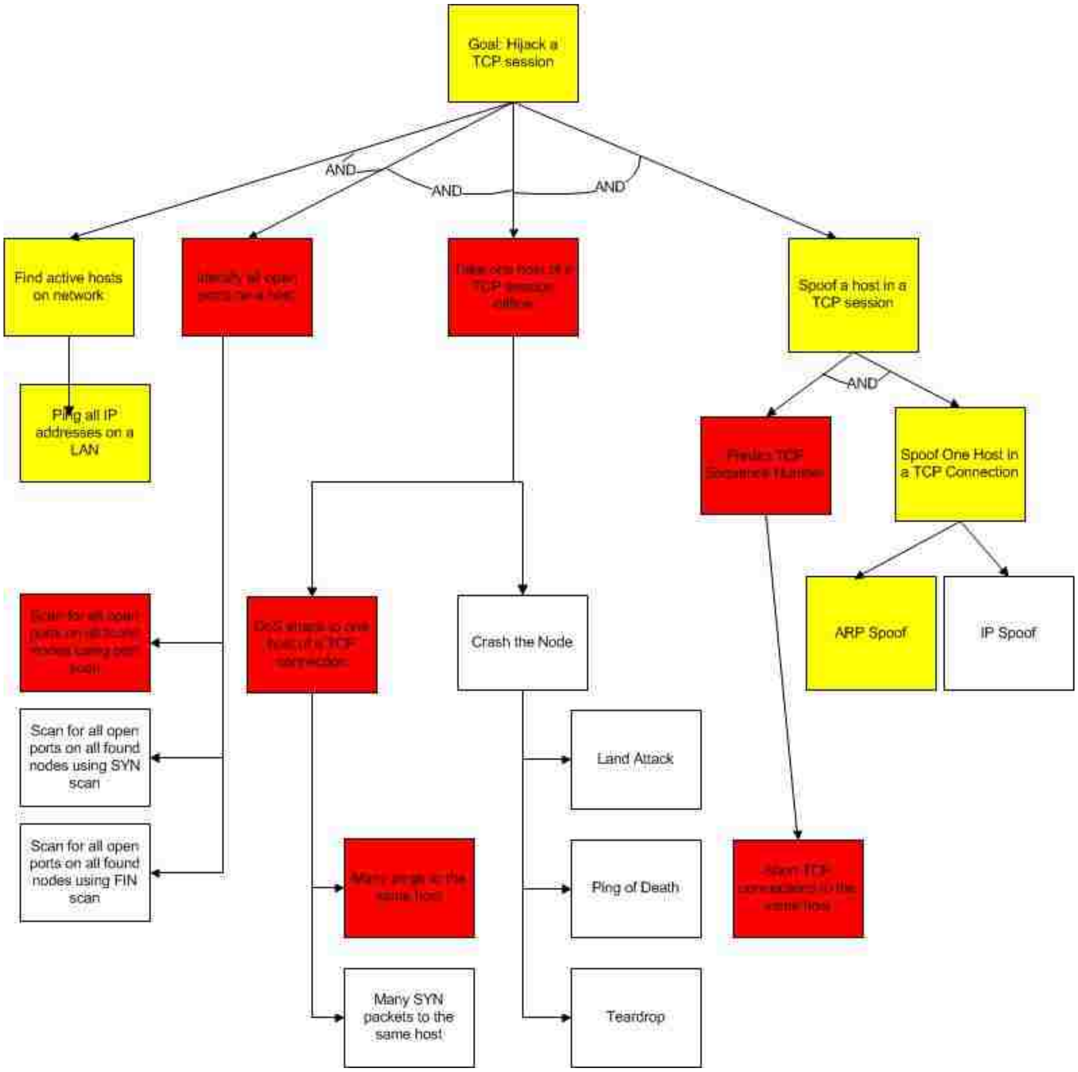

Goal: Hijack a TCP session
AND
AND
AND
Find active hosts on network
Identify all open ports on a host
Take one host of a TCP session offline
Spoof a host in a TCP session
AND
Ping all IP addresses on a LAN
Predict TCP Sequence Number
Spoof One Host in a TCP Connection
ARP Spoof
IP Spoof
Scan for all open ports on all found nodes using ping scan
DoS attack to one host of a TCP connection
Crash the Node
Scan for all open ports on all found nodes using SYN scan
Land Attack
Scan for all open ports on all found nodes using FIN scan
Many pings to the same host
Ping of Death
Abort TCP connections to the same host
Many SYN packets to the same host
Teardrop

Figure 8.2: An example of a colored attack tree.

Within each value for the rate, probability is used because the number of occurrences is still important. For instance, assume the threshold for the rate category is determined to be twenty-five. If there are fifty-one occurrences of a specific packet type, a value of 4 is assigned. If there are five hundred occurrences, a value of 4 is also assigned, but there is much more likelihood that there was a ping flood attack. The probability for the five hundred occurrences should be higher than for the fifty-one occurrences.

Probabilities will also be utilized as the colors are propagated up the tree. Instead of just using colors of the children nodes to color the parent node, the probabilities will be utilized. For instance, if a parent has two children nodes and they are both yellow, the current algorithm, shown in Fig. 8.3, colors the parent yellow. Using an algorithm that incorporates the probabilities in the children nodes may color the parent node red if the probabilities are high in both of the yellow children nodes.

```
OR conditions between children
•   Color parent the color of the child with the "largest" color

AND conditions between children
•   If all children are green → color parent green
•   Else (skip all green children)
    o   Find the color of the majority of the children (if the same
        number of yellow and red, then use color of latest child
        colored) →currColor
    o   If parent color <= currColor→
                         color parent currColor
        Else if parent colored more than "time ago" →
                         color parent currColor
        Else leave parent as-is
```

Figure 8.3: The current coloring scheme algorithm.

Probability will also be utilized in looking at the time frame for the attack. A lower probability will be assigned to the possibility of the attack occurring if the attack spans a longer time frame. If the attack spans a shorter time frame, there is more likelihood the attack actually occurred so a higher probability is assigned. This will be used in identifying simple and complex attacks.

For example, fifty ping packets to the same host in an hour may indicate a ping flood attack. Fifty ping packets to the same host in five seconds indicate that a ping flood attack most likely occurred, so the probability will be higher than the fifty in an hour attack. For complex attacks, the probability is higher if all the simple attacks comprising that complex attack occur in one hour compared to one day or one week, so again the shorter time frame indicates a higher probability.

8.2.2.4 Anomaly Detection

Anomaly detection is used in IDSs to detect the occurrence of an attack by observing behavior in the network that is unusual for that particular network and its users. For this to work effectively, normal behavior must be observed and documented. Future work will consist of incorporating anomaly detection into the formal representation of complex attacks.

The formal representation must be extended to maintain information about normal network traffic. The formal representation is then defined in OWL and incorporated into the set of heuristics developed for attack detection. This information (normal behavior) may also be useful in the *training phase* for determining threshold values based on network behavior.

8.2.2.5 Scalability Improvements

One drawback of the prototype system developed, TRIDSO, is its scalability. There has been research in the area of ontology scalability with several options for improvement. One method to improve on the scalability of TRIDSO is to distribute some of the processing. Concurrent execution of many of the SPARQL queries is one area of processing that may benefit from distribution. Goodman and Mizell [110] demonstrated the use of work-load distributions by developing an algorithm that utilized threads. The threads were used with replicated ontology data and a shared hash table. The second method that may benefit TRIDSO by providing more scalability is the use of a data management system. A data management system was developed specifically for OWL, by Park, et. al. [111], to "efficiently manage large sized OWL data" [111]. Park, et. al. increased the performance of queries by improving the management of large sized data sets. The performance improvement was achieved by storing the OWL data in a relational database designed to optimize query response. A third method to improve the scalability is to use a system that combines Datalog programs with a relational database. Pan, Li, and Heflin [112] developed such a system (DLDB3). DLDB3 is a new knowledge base system that showed an improvement of the system performance in load and query times.

Bibliography

[1] P. Spyns, R. Meersman, and M. Jarrar. Data modeling versus ontology engineering. *ACM SIGMOD Record*, 31(4): 12-17, 2002.

[2] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2): 93-136, 1996.

[3] T. R. Peltier. *Information security risk analysis*. Auerbach Publication, 2005.

[4] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC1157: A simple network management protocol (SNMP). 1990.

[5] M. Mealling. RFC3061: A URN namespace of object identifiers. 2001.

[6] J. E. López de Vergara, V. A. Villagrá, and J. Berrocal. Applying the web ontology language to management information definitions. *IEEE Communications Magazine, special issue on XML Management*, 42(7): 68-74, 2004.

[7] J. Hebeler, M. Fisher, R. Blace, and A. Perez-Lopez. *Semantic Web Programming*. Wiley, 2009.

[8] World Wide Web Consortium (W3C). W3C, 2011. Retrieved from http://www.w3.org/.

[9] World Wide Web Foundation. 2011. Retrieved from http://www.webfoundation.org/.

[10] T. Berners-Lee. Semantic Web - XML2000. *XML 2000 Proceedings*. Graphic Communications Association, December, 2000. Retrieved from http://www.w3.org/2000/Talks/1206-xml2k-tbl/.

[11] Extensible Markup Language (XML). World Wide Web Consortium (W3C), 2011. Retrieved from http://www.w3.org/XML/.

[12] Resource Description Framework (RDF). World Wide Web Consortium (W3C), 2010. Retrieved from http://www.w3.org/RDF/.

[13] RDF Vocabulary Description Language 1.0: RDF Schema. World Wide Web Consortium (W3C), 2004. Retrieved from http://www.w3.org/TR/rdf-schema/.

[14] G. Antoniou and F. van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, 2004.

[15] Web Ontology Language (OWL). World Wide Web Consortium (W3C), 2004. Retrieved from http://www.w3.org/2004/OWL/.

[16] L. W. Lacy. *Owl: Representing Information Using the Web Ontology Language*. Trafford Publishing, 2005.

[17] SPARQL Query Language for RDF. World Wide Web Consortium (W3C), 2008. Retrieved from http://www.w3.org/TR/rdf-sparql-query/.

[18] Jena - A Semantic Web Framework for Java. Retrieved from http://jena.sourceforge.net/index.html.

[19] G. Giambene. *Queuing Theory and Telecommunications: Networks and Applications*. Springer, 2010.

[20] D. Bertsekas and R. Gallager. *Data Networks*. Upper Saddle River, NJ, Prentice-Hall, Inc, 1992.

[21] D. Gross. *Fundamentals of Queueing Theory*. Wiley, 2009.

[22] L. Kleinrock. *Queueing Systems. Volume 1: Theory*. John Wiley & Sons, Inc., 1975.

[23] A. Fuchsberger. Intrusion detection systems and intrusion prevention systems. *Information Security Technical Report*. 10(3): 134-139, Jan. 2005.

[24] U. Warrier, L. Besaw, L. LaBarre, and B. Handspicker. RFC1189: The common management information services and protocols for the Internet (CMOT and CMIP). 1990.

[25] Desktop Management Interface (DMI). Distributed Management Task Force, Inc., 2010. Retrieved from http://dmtf.org/standards/dmi.

[26] Web-Based Enterprise Management (WBEM). Distributed Management Task Force, Inc., 2009. Retrieved from http://www.dmtf.org/standards/wbem/.

[27] W. Chen, N. Jain, and S. Singh. ANMP: Ad hoc network network management protocol. *IEEE Journal on Selected Areas in Communications*. 17: 1506-1531, 1999.

[28] L. B. Ruiz, J. M. Nogueira, and A. A. F. Loureiro. MANNA: A management architecture for wireless sensor networks. *IEEE Communications magazine*. 41(2): 116-125, 2003.

[29] G. Tolle, and D. Culler. Design of an application-cooperative management system for wireless sensor networks. *Proceedings of the Second European Workshop on Wireless Sensor Networks 2005*, 121-132, Istanbul, Turkey, 2005.

[30] A. K. Y. Wong, P. Ray, N. Parameswaran and J. Strassner. Ontology mapping for the interoperability problem in network management. *IEEE Journal on Selected Areas in Communications*, 23(10): 2058-2068, October 2005.

[31] J. E. López de Vergara, V. A. Villagrá, J. Berrocal, J. I. Asensio, and R. Pignaton. Semantic management: application of ontologies for the integration of management information models. *Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management*, Colorado Springs, CO, March 2003.

[32] J. E. López de Vergara, V. A. Villagrá, J. I. Asensio, and J. Berrocal. Ontologies: Giving semantics to network management models. *IEEE Network*, 17(3): 15-21, May-June 2003.

[33] J. E. López de Vergara, V. A. Villagrá, and J. Berrocal. An ontology-based method to merge and map management information models. *Proceedings of the HP Openview University Association Tenth Plenary Workshop*, Geneva, Switzerland. 2003.

[34] J. E. López de Vergara, V. A. Villagrá, and J. Berrocal. Applying the web ontology language to management information definitions. *IEEE Communications Magazine, special issue on XML Management*, 42(7): 68-74, July 2004.

[35] D. Cleary, B. Danev, and D. O'Donoghue. Using ontologies to simplify wireless network configuration. *Formal Ontologies Meet Industry*, Verona, Italy, June 2005.

[36] B. Deb, S. Bhatnagar and B. Nath. A topology discovery algorithm for sensor networks with applications to network management. Technical Report dcs-tr-441, Rutgers University, May 2001.

[37] M. E. Orwat, T. E. Levin, and C. E. Irvine. An ontological approach to secure MANET management. *Proceeedings of the Third International Conference on Availability, Reliability and Security*, pp. 787-794. 2008.

[38] C. Goodwin and D. J. Russomanno. An ontology-based sensor network prototype environment. *IEEE Fifth International Conference on Information Processing in Sensor Networks*, Nashville, TN, 2006.

[39] D. J. Russomanno, C. R. Kothari and O. A. Thomas. Building a sensor ontology: A practical approach leveraging ISO and OGC models. *The 2005 International Conference on Artificial Intelligence*, Las Vegas, NV, 2005.

[40] H. Honkasalo, K. Pehkonen, M. T. Niemi, and A. T. Leino. WCDMA and WLAN for 3G and beyond. *IEEE Wireless Communications*, 9(2), 2002.

[41] I. Niles and A. Pease, Origins of the standard upper merged ontology: A proposal for the IEEE standard upper ontology. Working Notes of the *IJCAI-2001 Workshop on the IEEE Standard Upper Ontology*, Seattle, WA, 2001.

[42] SensorML. Open Geospatial Consortium, 2011. Retrieved from http://www.opengeospatial.org/standards/sensorml.

[43] Crossbow Technology. Retrieved from http://www.xbow.com/.

[44] M N. Ismail and A. M. Zin. Development of simulation model in heterogeneous network environment: Comparing the accuracy of simulation model for data transfers measurement over wide area network. *International Journal of Multimedia and Ubiquitous Engineering*, 5(4): 43-58, 2010.

[45] M. Hedayati, S. H. Kamali, and A. S. Izadi. The monitoring of the network traffic based on queuing theory and simulation in heterogeneous network environment. *Proceedings of the 2009 International Conference on Information and Multimedia Technology (ICIMT '09)*, pp. 396-402, Jeju Island, South Korea**,** 2009.

[46] M.-Y. Huang and T. M. Wicks, A large-scale distributed intrusion detection framework based on attack strategy analysis. *Second International Workshop on Recent Advances in Intrusion Detection, RAID'98*, Louvain-la-Neuve, Belgium, September 1998.

[47] S. A. Camtepe and B. Yener, Modeling and detection of complex attacks. *Third International Conference on Security and Privacy in Communications Networks and the Workshops (SecureComm 2007)*, pp. 234-243, Nice, France, September 2007.

[48] I. Gregorio-deSouza, V. H. Berk, A. Giani, G. Bakos, M. Bates, G. Cybenko, and D. Madory. Detection of complex cyber attacks. *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V*, May 2006.

[49] Snort. 2010. Retrieved from http://www.snort.org/.

[50] B. Caswell, J. Beale, and A. Baker. *Snort IDS and IPS Toolkit*. Burlington, MA, Syngress Publishing, Inc., 2007.

[51] S. Sen. Performance characterization and improvement of intrusion detection systems. Bell Labs, Alcatel-Lucent, 2006.

[52] X. Ou, S. Govindavajhala, A. Appel. MulVAL: A logic-based network security analyzer. *14th USENIX Security Symposium*, Baltimore, MD, August 2005.

[53] H. Xu, D. Xiao, and Z. Wu. Application of security ontology to context-aware alert analysis. *Eighth IEEE/ACIS International Conference on Computer and Information Science (ICIS 2009)*, pp. 171-176, Shanghai, 2009.

[54] L. A. F. Martimiano and E. Moreira. The evaluation process of a computer security incident ontology. *The 2nd Workshop on Ontologies and their Applications (WONTO'06)*, Ribeirão Preto, SP, Brazil, 2006.

[55] L. A. F. Martimiano and E. d. S. Moreira. An OWL-based security incident ontology (a poster session). *Eighth International Protégé Conference*, Madrid, Spain, 2005.

[56] B. Tsoumas and D. Gritzalis. Towards an ontology-based security management. *20th International Conference on Advanced Information Networking and Applications (AINA'06)*, 1: 985-992, Vienna, Austria.2006.

[57] N. Cuppens-Boulahia, F. Cuppens, J. E. López de Vergara, E. Vazquez, J. Guerra, and H. Debar. An ontology-based approach to react to network attacks. *International Journal of Information and Computer Security*, 3(3/4): 280-305, 2009.

[58] A. Vorobiev and N. Bekmamedova. An ontological approach applied to information security and trust. *18th Australasian Conference on Information Systems (ACIS 2007)*, Toowoomba, Queensland, Australia, 2007.

[59] A. Vorobiev and J. Han. Security attack ontology for web services. *Second International Conference on Semantics, Knowledge and Grid (SKG '06)*, Guilin, China, 2006.

[60] A. Vorobiev, J. Han, and N. Bekmamedova. An ontology framework for managing security attacks and defences in component based software systems. *19th Australian Conference on Software Engineering (ASWEC 2008)*, pp. 552-561, Perth, WA, 2008.

[61] J. Undercoffer, A. Joshi, and J. Pinkston. Modeling computer attacks: An ontology for intrusion detection. *The Sixth International Symposium on Recent Advances in Intrusion Detection*, G. Vigna, E. Jonsson and C. Kruegel, editors, Springer, LNCS 2820: 113-135, 2003.

[62] J. Undercoffer and J. Pinkston. Modeling computer attacks: A target-centric ontology for intrusion detection. *2002 CADIP Research Symposium University of Maryland Baltimore County (UMBC)*, Baltimore, MD, 2002.

[63] S. Mandujano. An ontology-supported outbound intrusion detection system. *Proceedings of the 10th Conference on Artificial Intelligence and Applications, Taiwanese Association for Artificial Intelligence (TAAI 2005)*, Kaohsiung, Taiwan, December 2005.

[64] S. Mandujano, A. Galvan, and J. A. Nolazco. An ontology-based multiagent approach to outbound intrusion detection. *ACS/IEEE 2005 International Conference on Computer Systems and Applications (AICCSA'05)*, Cairo, Egypt, January 2005.

[65] OVAL (Open Vulnerability and Assessment Language). The MITRE Corporation, 2011. Retrieved from http://oval.mitre.org/.

[66] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1: 146–166, March 1989.

[67] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language combining OWL and RuleML. National Research Council of Canada, Network Inference, and Stanford University, 2004. Retrieved from http://www.w3.org/Submission/SWRL/.

[68] CIM Schema: Version 2.8.2 (Final). Distributed Management Task Force, Inc., 2010. Retrieved from http://dmtf.org/standards/cim/cim_schema_v282.

[69] CVE: Common Vulnerabilities and Exposures. The MITRE Corporation, 2011. Retrieved from http://cve.mitre.org/.

[70] H. Debar, D. Curry, and B. Feinstein. RFC4765: The Intrusion Detection Message Exchange Format (IDMEF). 2007.

[71] OrBAC. 2008. Retrieved from http://orbac.org/index.php?page=orbac&lang=en.

[72] National Vulnerability Database. National Institute of Standards and Technology (NIST), 2011. Retrieved from http://nvd.nist.gov/.

[73] ProIT - overview. PerformanceIT, Inc., 2011. Retrieved from http://www.performanceit.com/proit_overview.html.

[74] L. Frye and L. Cheng. A network management system for a heterogeneous multi-tier network. *IEEE Global Communications Conference, Exhibition and Industry Forum (GLOBECOM)*, Miami, FL. December, 2010.

[75] FaCT++. Google, 2011. Retrieved from http://code.google.com/p/factplusplus/.

[76] K. Fox. *Ad-hoc network management: Using the Simple Network Management Protocol in an ad-hoc environment*. Master thesis, Kutztown University, Kutztown, PA. 2010.

[77] B. Deb and B. Nath. Wireless sensor networks management. 2005. Retrieved from http://www.research.rutgers.edu/_bdeb/sensornetworks.html.

[78] W. L. Lee, A. Datta, and R. Cardell-Oliver. Network management in wireless sensor networks. In L. T. Yang and M. K. Denko, editors, *Handbook on Mobile Ad Hoc and Pervasive Communications*, American Scientific Publishers, 2006.

[79] M. Fernández, M., A. Gómez-Pérez, and N. Juristo. Methontology: From ontological art towards ontological engineering. *AAAI Technical Report***,** pp. 33-40, 1997.

[80] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 1993.

[81] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. *Technical Report*. Stanford University, Stanford, CA, 2001.

[82] M. Daniele, B. Wijnen, M. Ellison, and D. Francisco. RFC2741: Agent Extensibility (AgentX) Protocol. 2001.

[83] Net-SNMP. 2011. Retrieved from http://www.net-snmp.org/.

[84] Nishida, T. End-to-End performance modeling for distributed network management systems. *Proceedings of the Tenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) 1991*, pp. 121-129, Bal Harbour, FL, 1991.

[85] L. Frye, Z. Liang, K. Fox, and L. Cheng. An automated network management system for heterogeneous multi-tier networks. *Journal of Network and System Management*. Springer. Unpublished Work.

[86] D. Bertsekas and R. Gallager. *Data Networks*, chapter 4, pp.318. Prentice Hall, 1987.

[87] S. S. Lam. A carrier sense multiple access protocol for local networks. *Computer Networks,* 4(1): 21-32, February 1980.

[88] W. Stallings. *Operating Systems: Internals and Design Principles*, 3rd Edition. Prentice Hall Engineering/Science/Mathematics, December 1997.

[89] Nmap Security Scanner. Retrieved from http://nmap.org/.

[90] SING. Geeknet, Inc., 2011. Retrieved from http://sourceforge.net/projects/sing/.

[91] L. Frye, L. Cheng, and R. Kaplan. A methodology to identify complex network attacks. *The 2011 International Conference on Security and Management (SAM'11) at the 2011 World Congress in Computer Science*, *Computer Engineering, and Applied Computing (WORLDCOMP'11)*, Las Vegas, NV, July 2011.

[92] Information Assurance (IA). *Directive 8500.01E*. Department of Defense, 2002.

[93] C.W. Geib and R. Goldman. Plan Recognition in Intrusion Detection Systems. *Proceedings of the Second DARPA Information Survivability Conference and Exposition (DISCEX II)*, pp. 329-342, 2001.

[94] C. W. Geib. Assessing the complexity of plan recognition. *9th national conference on Artifical intelligence (AAAI'04)*, pp. 507-512, San Jose, CA, July 2004.

[95] T. Kichkaylo, T. Ryutov, M. D. Orosz, and R. Neches. Planning to Discover and Counteract Attacks. *Informatica*, 34: 159-168. 2010.

[96] C. W. Geib. Toward Using Plan Recognition for Intrusion Detection. *Proceedings of the ICAPS Workshop on Intelligent Security*, pp. 46-55, 2009.

[97] L. Frye, L. Cheng, and J. Heflin. An ontology-based system to identify complex network attacks. *IEEE/IFIP Network Operations and Management Symposium (NOMS 2012)*, Maui, HI, April 2012. Unpublished Work

[98] Wireshark. Retrieved from http://www.wireshark.org/.

[99] L. Chappell and G. Combs. *Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide*. Laura Chappell University, 2010.

[100]J. A. Wang and M. Guo. OVM: An Ontology for Vulnerability Management. *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW): Cyber Security and Information Intelligence Challenges and Strategies*, Oak Ridge, TN, ACM, 2009.

[101]The Network Vulnerability Scanner. Tenable Network Security, 2011. Retrieved from http://www.nessus.org/nessus/intro.php.

[102]SSA - Security System Analyzer. Security Database. Retrieved from http://www.security-database.com/ssa.php.

[103] L. Obrst, B. Ashpole, W. Ceusters, I. Mani, S. Ray and B. Smith. The evaluation of ontologies: Toward improved semantic interoperability. In C. J. O. Baker and K.-H. Cheung, editors, *Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences*, pp. 139-158, Springer, 2007.

[104] tcpdump. 2010. Retrieved from http://www.tcpdump.org/.

[105] Ettercap. Retrieved from http://ettercap.sourceforge.net/.

[106] Jena 2 Inference Support - A Semantic Web Framework for Java. 2010. Retrieved from http://jena.sourceforge.net/inference/index.html.

[107] H. Bohring and S. Auer. Mapping XML to OWL ontologies. In *Leipziger Informatik-Tage, volume 72 of Lecture Notes in Informatics (LNI)*, 72: 147-156, 2005.

[108] Metadata Management: Providing Static Metadata of Robotic Telescopes to Stellaris (an AstroGrid-D Deliverable). Part of the AstroGrid-D German Astronomy Community Grid (GACG) project funded by the German Federal Ministry of Education and Research (BMBF), 2007. Retrieved from http://www.gac-grid.org/project-documents/deliverables/wp2/D2_4.pdf.

[109] The Extensible Stylesheet Language Family (XSL). World Wide Web Consortium (W3C), 2011. Retrieved from http://www.w3.org/Style/XSL/.

[110] E. L. Goodman and D. Mizell. Scalable in-memory RDFS closure on billions of triples. *The 6th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2010)*, pp. 17-31, Shanghai, China**,** November 2010.

[111] M.-J. Park, J. Lee, C.-H. Lee, J. Lin, O. Serres, and C.-W. Chung. An efficient and scalable management of ontology. *International Conference of Database Systems for Advanced Applications*, pp. 975-98, Bangkok, Thailand**,** April 2007.

[112] Z. Pan, Y. Li, and J. Heflin. A semantic web knowledge base system that supports large scale data integration. *8th International Semantic Web Conference (ISWC2009)*, pp**.** 125-140, Washington DC**,** October 2009.

Appendix A   A Heterogeneous Multi-tier Network Management System - Ontology Definition Files

A.1 Node Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://faculty.kutztown.edu/frye/res/onto/node.owl#"
    xml:base="http://faculty.kutztown.edu/frye/res/onto/node.owl">

  <owl:Ontology rdf:about=""/>


  <!-- Create a class for a network node -->
  <owl:Class rdf:ID="node"/>

  <owl:DatatypeProperty rdf:ID="serialNumber">
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="name">
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="location">
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="address">
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="description">
  </owl:DatatypeProperty>


</rdf:RDF>
```

A.2 Wired Node Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
   <!ENTITY g 'http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#'>
   <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns="http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl">

  <owl:Ontology rdf:about="">
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
   <owl:imports
          rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl"/>
  </owl:Ontology>


  <!-- Create a class for a network wired node -->
  <owl:Class rdf:ID="wiredNode">
      <rdfs:subClassOf rdf:resource="&e;node"/>
      <owl:disjointWith rdf:resource="&g;wirelessNode"/>
  </owl:Class>


  <owl:DatatypeProperty rdf:ID="subnetMask">
  </owl:DatatypeProperty>

</rdf:RDF>
```

A.3 Wireless Node Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
    <!ENTITY f 'http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#'>
    <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#"
    xml:base="http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl">

  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
    <owl:imports
                        rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl"/>
  </owl:Ontology>

  <!-- Class for node's role -->
  <owl:Class rdf:ID="roleType"/>
  <roleType rdf:ID="ch"/>

  <!-- Class for node's status -->
  <owl:Class rdf:ID="statusType"/>

  <!-- Create a class for a network wireless node -->
  <owl:Class rdf:ID="wirelessNode">
            <rdfs:subClassOf rdf:resource="&e;node"/>
            <owl:disjointWith rdf:resource="&f;wiredNode"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="energyLeft">
  </owl:DatatypeProperty>

  <owl:ObjectProperty rdf:ID="role">
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="status">
  </owl:ObjectProperty>
</rdf:RDF>
```

A.4 Nortel Device Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY c 'http://faculty.kutztown.edu/frye/res/onto/cisco.owl#'>
   <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
   <!ENTITY f 'http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#'>
   <!ENTITY g 'http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#'>
  <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns="http://faculty.kutztown.edu/frye/res/onto/nortel.owl#"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/nortel.owl">

<owl:Ontology rdf:about="">
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
   <owl:imports
         rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl"/>
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/cisco.owl"/>
</owl:Ontology>

  <owl:Class rdf:ID="nortelNode">
         <rdfs:subClassOf rdf:resource="&f;wiredNode"/>
         <owl:disjointWith rdf:resource="&c;ciscoNode"/>
 </owl:Class>


  <owl:DatatypeProperty rdf:ID="rcSysIPAddr">
   <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="sysDesc">
   <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="sysName">
   <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="sysLocation">
   <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="rcChasSerialNumber">
```

```
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="sysNetMask">
  </owl:DatatypeProperty>

</rdf:RDF>
```

A.5 Cisco Device Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY d 'http://faculty.kutztown.edu/frye/res/onto/nortel.owl#'>
   <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
   <!ENTITY f 'http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#'>
   <!ENTITY g 'http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#'>
  <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns="http://faculty.kutztown.edu/frye/res/onto/cisco.owl#"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/cisco.owl">

<owl:Ontology rdf:about="">
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
   <owl:imports
         rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl"/>
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/nortel.owl"/>
</owl:Ontology>

 <owl:Class rdf:ID="ciscoNode">
         <rdfs:subClassOf rdf:resource="&f;wiredNode"/>
         <owl:disjointWith rdf:resource="&d;nortelNode"/>
 </owl:Class>


 <owl:DatatypeProperty rdf:ID="chassisSerialNumber">
   <rdfs:domain rdf:resource="&e;node"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="sysName">
   <rdfs:domain rdf:resource="&e;node"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="sysLocation">
   <rdfs:domain rdf:resource="&e;node"/>
 </owl:DatatypeProperty>


 <owl:DatatypeProperty rdf:ID="sysNetMask">
 </owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:ID="sysIPAddr">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="sysDesc">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

A.6 Ad hoc Device Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY b 'http://faculty.kutztown.edu/frye/res/onto/wsn.owl#'>
    <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
    <!ENTITY f 'http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#'>
    <!ENTITY g 'http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#'>
    <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://faculty.kutztown.edu/frye/res/onto/adhoc.owl#"
    xmlns:wireless="&g;"
    xml:base="http://faculty.kutztown.edu/frye/res/onto/adhoc.owl">

  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
    <owl:imports
            rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl"/>
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wsn.owl"/>
  </owl:Ontology>


  <!-- object instances for status and role -->
  <wireless:statusType rdf:ID="active"/>
  <wireless:statusType rdf:ID="not_active"/>
  <wireless:roleType rdf:ID="agent"/>


  <owl:Class rdf:ID="adHocNode">
            <rdfs:subClassOf rdf:resource="&g;wirelessNode"/>
            <owl:disjointWith rdf:resource="&b;sensor"/>
  </owl:Class>


  <owl:ObjectProperty rdf:ID="clusterHead">
    <rdfs:domain rdf:resource="&g;wirelessNode"/>
    <rdfs:range rdf:resource="#adHocNode"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="description">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="location">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="name">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>


  <owl:DatatypeProperty rdf:ID="ipAddress">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="subnetMask">
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="serialNumber">
    <rdfs:domain rdf:resource="&e;node"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="remainingBatteryLife">
    <rdfs:domain rdf:resource="&g;wirelessNode"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="role">
    <rdfs:domain rdf:resource="&g;wirelessNode"/>
    <rdfs:range rdf:resource="&g;roleType"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Role of ad hoc node, is it a CH or agent (plain) node</rdfs:comment>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="status">
    <rdfs:domain rdf:resource="&g;wirelessNode"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Status of the ad hoc node, is it active or inactive</rdfs:comment>
    <rdfs:range rdf:resource="&g;statusType"/>
 </owl:ObjectProperty>


</rdf:RDF>
```

A.7 Wireless Sensor Network Device Ontology Definition File

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY a 'http://faculty.kutztown.edu/frye/res/onto/adhoc.owl#'>
   <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
   <!ENTITY f 'http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#'>
   <!ENTITY g 'http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#'>
   <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns="http://faculty.kutztown.edu/frye/res/onto/wsn.owl#"
   xmlns:wireless="&g;"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/wsn.owl">

  <owl:Ontology rdf:about="">
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
   <owl:imports
          rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl"/>
   <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/adhoc.owl"/>
  </owl:Ontology>


  <!-- object instances for status and role -->
  <wireless:statusType rdf:ID="alive"/>
  <wireless:statusType rdf:ID="dead"/>
  <wireless:roleType rdf:ID="member"/>


  <owl:Class rdf:ID="sensor">
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >
          a node in a wireless sensor network</rdfs:comment>
     <rdfs:subClassOf rdf:resource="&g;wirelessNode"/>
     <owl:disjointWith rdf:resource="&a;adHocNode"/>
 </owl:Class>

  <owl:ObjectProperty rdf:ID="clusterHead">
   <rdfs:domain rdf:resource="&g;wirelessNode"/>
   <rdfs:range rdf:resource="#sensor"/>
  </owl:ObjectProperty>
```

```
<owl:DatatypeProperty rdf:ID="description">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Description for sensor</rdfs:comment>
  <rdfs:domain rdf:resource="&e;node"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ycoord">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        y-coordinate for sensor</rdfs:comment>
  <rdfs:domain rdf:resource="#sensor"/>
</owl:DatatypeProperty>



<owl:DatatypeProperty rdf:ID="residualEnergy">
  <rdfs:domain rdf:resource="&g;wirelessNode"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Residual energy of sensor</rdfs:comment>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="role">
  <rdfs:domain rdf:resource="&g;wirelessNode"/>
  <rdfs:range rdf:resource="&g;roleType"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Role of sensor, is it a CH, member or plain node</rdfs:comment>
</owl:ObjectProperty >
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Name for the sensor</rdfs:comment>
  <rdfs:domain rdf:resource="&e;node"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serialNumber">
  <rdfs:domain rdf:resource="&e;node"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        serial number of sensor</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="nodeID">
  <rdfs:domain rdf:resource="#sensor"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        node ID for sensor</rdfs:comment>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="status">
  <rdfs:domain rdf:resource="&g;wirelessNode"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Status of the sensor, is it alive or dead</rdfs:comment>
  <rdfs:range rdf:resource="&g;statusType"/>
</owl:ObjectProperty >
```

```
  <owl:DatatypeProperty rdf:ID="xcoord">
    <rdfs:domain rdf:resource="#sensor"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            x-coordinate for sensor</rdfs:comment>
  </owl:DatatypeProperty>


</rdf:RDF>
```

A.8 Mapping Ontology Definition File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY a 'http://faculty.kutztown.edu/frye/res/onto/adhoc.owl#'>
   <!ENTITY b 'http://faculty.kutztown.edu/frye/res/onto/wsn.owl#'>
   <!ENTITY c 'http://faculty.kutztown.edu/frye/res/onto/cisco.owl#'>
   <!ENTITY d 'http://faculty.kutztown.edu/frye/res/onto/nortel.owl#'>
   <!ENTITY e 'http://faculty.kutztown.edu/frye/res/onto/node.owl#'>
   <!ENTITY f 'http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl#'>
   <!ENTITY g 'http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl#'>
  <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:adhoc="&a;"
   xmlns:wsn="&b;"
   xmlns:cisco="&c;"
   xmlns:nortel="&d;"
   xmlns:node="&e;"
   xmlns:wired="&f;"
   xmlns:wireless="&g;"
   xmlns="http://faculty.kutztown.edu/frye/res/onto/map_all.owl#"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/map_all.owl">

 <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/node.owl"/>
    <owl:imports
           rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wiredNode.owl"/>
    <owl:imports
           rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wirelessNode.owl"/>
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/adhoc.owl"/>
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wsn.owl"/>
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/cisco.owl"/>
    <owl:imports rdf:resource="http://faculty.kutztown.edu/frye/res/onto/nortel.owl"/>
    <owl:imports
           rdf:resource="http://faculty.kutztown.edu/frye/res/onto/adhoc_instances.owl"/>
    <owl:imports
           rdf:resource="http://faculty.kutztown.edu/frye/res/onto/wsn_instances.owl"/>
    <owl:imports
           rdf:resource="http://faculty.kutztown.edu/frye/res/onto/cisco_instances.owl"/>
    <owl:imports
           rdf:resource="http://faculty.kutztown.edu/frye/res/onto/nortel_instances.owl"/>
 </owl:Ontology>


 <owl:DatatypeProperty rdf:about="&e;name">
  <owl:equivalentProperty rdf:resource="&a;name"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;name">
  <owl:equivalentProperty rdf:resource="&b;name"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;name">
  <owl:equivalentProperty rdf:resource="&c;sysName"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;name">
  <owl:equivalentProperty rdf:resource="&d;sysName"/>
 </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:about="&e;description">
  <owl:equivalentProperty rdf:resource="&a;description"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;description">
  <owl:equivalentProperty rdf:resource="&b;description"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;description">
  <owl:equivalentProperty rdf:resource="&c;sysDesc"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;description">
  <owl:equivalentProperty rdf:resource="&d;sysDesc"/>
 </owl:DatatypeProperty>

 <owl:DatatypeProperty rdf:about="&e;serialNumber">
  <owl:equivalentProperty rdf:resource="&a;serialNumber"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;serialNumber">
  <owl:equivalentProperty rdf:resource="&b;serialNumber"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;serialNumber">
  <owl:equivalentProperty rdf:resource="&c;chassisSerialNumber"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&e;serialNumber">
  <owl:equivalentProperty rdf:resource="&d;rcChasSerialNumber"/>
 </owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:about="&a;ipAddress">
  <rdfs:subPropertyOf rdf:resource="&e;address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&b;nodeID">
  <rdfs:subPropertyOf rdf:resource="&e;address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&c;sysIPAddr">
  <rdfs:subPropertyOf rdf:resource="&e;address"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&d;rcSysIPAddr">
  <rdfs:subPropertyOf rdf:resource="&e;address"/>
</owl:DatatypeProperty>


<owl:DatatypeProperty rdf:about="&f;subnetMask">
  <owl:equivalentProperty rdf:resource="&a;subnetMask"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&f;subnetMask">
  <owl:equivalentProperty rdf:resource="&c;sysNetMask"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&f;subnetMask">
  <owl:equivalentProperty rdf:resource="&d;sysNetMask"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&a;location">
  <rdfs:subPropertyOf rdf:resource="&e;location"/>
</owl:DatatypeProperty>


<owl:DatatypeProperty rdf:about="&b;xcoord">
  <rdfs:subPropertyOf rdf:resource="&e;location"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&b;ycoord">
  <rdfs:subPropertyOf rdf:resource="&e;location"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&c;sysLocation">
  <rdfs:subPropertyOf rdf:resource="&e;location"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="&d;sysLocation">
  <rdfs:subPropertyOf rdf:resource="&e;location"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:about="&g;role">
  <owl:equivalentProperty rdf:resource="&a;role"/>
</owl:ObjectProperty >
```

```
 <owl:ObjectProperty rdf:about="&g;role">
   <owl:equivalentProperty rdf:resource="&b;role"/>
 </owl:ObjectProperty >

 <owl:ObjectProperty rdf:about="&g;status">
   <owl:equivalentProperty rdf:resource="&a;status"/>
 </owl:ObjectProperty >
 <owl:ObjectProperty rdf:about="&g;status">
   <owl:equivalentProperty rdf:resource="&b;status"/>
 </owl:ObjectProperty >


 <owl:DatatypeProperty rdf:about="&g;energyLeft">
   <owl:equivalentProperty rdf:resource="&a;remainingBatteryLife"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:about="&g;energyLeft">
   <owl:equivalentProperty rdf:resource="&b;residualEnergy"/>
 </owl:DatatypeProperty>


 <!-- Create a class for all  cluster heads -->
 <owl:Class rdf:ID="clusterHeadNode">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
         <owl:onProperty rdf:resource="&a;role"/>
         <owl:hasValue rdf:resource="&g;ch"/>
      </owl:Restriction>
      <owl:Class rdf:about="&g;wirelessNode"/>
   </owl:intersectionOf>
 </owl:Class>


</rdf:RDF>
```

Appendix B Complex Attack Detection - Ontology Definition Files

B.1 Traffic Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY traffic
         'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl#'>
   <!ENTITY attack
         'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl#'>
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
   <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
   <!ENTITY owl11 "http://www.w3.org/2006/12/owl11#">
]>

<!--
 ************************************************************************
 ************************************************************************
 *****                  Traffic Ontology                  *****
 ************************************************************************
 ************************************************************************
-->


<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:owl11="http://www.w3.org/2006/12/owl11#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:traffic="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl#"
   xmlns:attack="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl#"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl">

  <owl:Ontology rdf:about="">
         <rdfs:comment>An ontology for network traffic</rdfs:comment>
  </owl:Ontology>
```

```
<!--
*************************************************************************
*************************************************************************
*****              Object Property Definitions              *****
*************************************************************************
*************************************************************************
-->

<!--
*****           Object Properties: MAC addresses            *****
-->
<owl:ObjectProperty rdf:ID="hasSrcMAC">
        <rdfs:range rdf:resource="#MACaddress"/>
        <rdfs:domain rdf:resource="#L2Packet"/>
        <owl:inverseOf rdf:resource="#isSrcMACOf"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasSrcMac"/>

<owl:ObjectProperty rdf:ID="hasDestMAC">
        <rdfs:range rdf:resource="#MACaddress"/>
        <rdfs:domain rdf:resource="#L2Packet"/>
        <owl:inverseOf rdf:resource="#isDestMACOf"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasDestMAC"/>

<owl:ObjectProperty rdf:ID="isSrcMACOf">
        <rdfs:range rdf:resource="#L2Packet"/>
        <rdfs:domain rdf:resource="#MACaddress"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isDestMACOf">
        <rdfs:range rdf:resource="#L2Packet"/>
        <rdfs:domain rdf:resource="#MACaddress"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasNode1MAC">
        <rdfs:range rdf:resource="#MACaddress"/>
        <rdfs:domain rdf:resource="#L2Stream"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasNode1MAC"/>

<owl:ObjectProperty rdf:ID="hasNode2MAC">
        <rdfs:range rdf:resource="#MACaddress"/>
        <rdfs:domain rdf:resource="#L2Stream"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasNode2MAC"/>
```

```
<!--   *****            Object Properties: IP addresses                *****  -->
<owl:ObjectProperty rdf:ID="hasNWIPaddress">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasSrcIP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#IPPacket"/>
        <owl:inverseOf rdf:resource="#isSrcIPOf"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasSrcIP"/>

<owl:ObjectProperty rdf:ID="hasDestIP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#IPPacket"/>
        <owl:inverseOf rdf:resource="#isDestIPOf"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasDestIP"/>

<owl:ObjectProperty rdf:ID="isSrcIPOf">
        <rdfs:range rdf:resource="#IPPacket"/>
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isDestIPOf">
        <rdfs:range rdf:resource="#IPPacket"/>
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasPCDestIP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#PacketCollection"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasPCDestIP"/>

<owl:ObjectProperty rdf:ID="hasAlertSrcIP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#IPAlert"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasAlertSrcIP"/>
```

```
<owl:ObjectProperty rdf:ID="hasAlertDestIP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#IPAlert"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasAlertDestIP"/>

<owl:ObjectProperty rdf:ID="hasNode1IP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#L3Stream"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasNode1IP"/>

<owl:ObjectProperty rdf:ID="hasNode2IP">
        <rdfs:range rdf:resource="#IPaddress"/>
        <rdfs:domain rdf:resource="#L3Stream"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:about="#hasNode2IP"/>

<!-- Make 'hasTCPStreamWith' a Symmetric property, meaning it will hold in both
        directions. If, host A has a TCP stream with host B, then host B has a
        TCP stream with host A -->
<owl:SymmetricProperty rdf:ID="hasTCPStreamWith">
        <rdfs:domain rdf:resource="#IPaddress"/>
         <rdfs:range  rdf:resource="#IPaddress"/>
</owl:SymmetricProperty>



<!--
************************************************************************
************************************************************************
*****              Address Class Definitions              *****
************************************************************************
************************************************************************
-->

<!--   *****              Class: MACaddres              *****  -->
<owl:Class rdf:ID="MACaddress">
</owl:Class>

<!--    *****              Class: IPaddress              *****  -->
<owl:Class rdf:ID="IPaddress">
</owl:Class>
```

```
<owl:DatatypeProperty rdf:ID="IPoctet1">
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="IPoctet2">
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="IPoctet3">
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="IPoctet4">
        <rdfs:domain rdf:resource="#IPaddress"/>
</owl:DatatypeProperty>


<!--
*****************************************************************************
*****             Class: NWaddressScanned                         *****
*****************************************************************************
-->
<owl:Class rdf:ID="NWaddressScanned">
        <rdfs:comment>
          A list of Network IP addresses that were scanned with a PingScan
        </rdfs:comment>

        <rdfs:subClassOf rdf:resource="#IPaddress"/>
</owl:Class>



<!--
*****************************************************************************
*****************************************************************************
*****           Packet and related classes                        *****
*****************************************************************************
*****************************************************************************
-->

<!--
*****************************************************************************
*****              Class: Packet                            *****
*****************************************************************************
-->
<owl:Class rdf:ID="Packet">
</owl:Class>
```

```
<owl:DatatypeProperty rdf:ID="packetID">
        <rdfs:domain rdf:resource="#Packet"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="dateTime">
        <rdfs:domain rdf:resource="#Packet"/>
</owl:DatatypeProperty>

<owl:Restriction>
  <owl:onProperty rdf:resource="#packetID" />
  <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction>



<!--
*************************************************************************
*****             Class: L2Packet                       *****
*************************************************************************
-->
<owl:Class rdf:ID="L2Packet">
        <rdfs:subClassOf rdf:resource="#Packet"/>
</owl:Class>



<!--
*************************************************************************
*****             Class: IPPacket                       *****
*************************************************************************
-->
<owl:Class rdf:ID="IPPacket">
        <rdfs:subClassOf rdf:resource="#L2Packet"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="ver">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hdrLen">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="packetLen">
         <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="transProto">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="flags">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="fragment">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="fragOffset">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ttl">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ipChecksum">
        <rdfs:domain rdf:resource="#IPPacket"/>
</owl:DatatypeProperty>


<!--
*************************************************************************
*****              Class: L4Packet                        *****
*************************************************************************
-->
<owl:Class rdf:ID="L4Packet">
        <rdfs:subClassOf rdf:resource="#IPPacket"/>
        <owl:disjointWith rdf:resource="#ICMPPacket"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="l4SrcPort">
        <rdfs:domain rdf:resource="#L4Packet"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="l4DestPort">
        <rdfs:domain rdf:resource="#L4Packet"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="l4Checksum">
         <rdfs:domain rdf:resource="#L4Packet"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="l4Payload">
        <rdfs:domain rdf:resource="#L4Packet"/>
        <rdfs:range rdf:resource="#Application"/>
</owl:ObjectProperty>
```

```
<!--
*************************************************************************
*****              Class: TCPPacket                         *****
*************************************************************************
-->
<owl:Class rdf:ID="TCPPacket">
        <rdfs:subClassOf rdf:resource="#L4Packet"/>
        <owl:disjointWith rdf:resource="#UDPPacket"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="tcpSeqNum">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpAckNum">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpFlags">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpAckFlag">
         <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpRstFlag">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpSynFlag">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpFinFlag">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="tcpWinSize">
        <rdfs:domain rdf:resource="#TCPPacket"/>
</owl:DatatypeProperty>


<!--
*************************************************************************
*****              Class: UDPPacket                         *****
*************************************************************************
-->
<owl:Class rdf:ID="UDPPacket">
        <rdfs:subClassOf rdf:resource="#L4Packet"/>
        <owl:disjointWith rdf:resource="#TCPPacket"/>
</owl:Class>
```

```
 <!--
************************************************************************
*****            Class: AppPacket                    *****
*****   This class is the union of the TCPPacket class          *****
*****    and the UDPPacket class.                          *****
************************************************************************
-->
<owl:Class rdf:ID="AppPacket">
        <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#TCPPacket"/>
                <owl:Class rdf:about="#UDPPacket"/>
        </owl:unionOf>
</owl:Class>


<!--
************************************************************************
*****            Class: ICMPPacket                   *****
************************************************************************
-->
<owl:Class rdf:ID="ICMPPacket">
        <rdfs:subClassOf rdf:resource="#IPPacket"/>
        <owl:disjointWith rdf:resource="#L4Packet"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="icmpType">
        <rdfs:domain rdf:resource="#ICMPPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="icmpCode">
        <rdfs:domain rdf:resource="#ICMPPacket"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="icmpPayload">
        <rdfs:domain rdf:resource="#UDPPacket"/>
        <rdfs:range rdf:resource="#Application"/>
</owl:ObjectProperty>


<!--
************************************************************************
*****             Class: Application                 *****
************************************************************************
-->
<owl:Class rdf:ID="Application">
</owl:Class>
```

```
<owl:DatatypeProperty rdf:ID="appProtocol">
	<rdfs:domain rdf:resource="#Application"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="appData">
	<rdfs:domain rdf:resource="#Application"/>
</owl:DatatypeProperty>



<!--
*******************************************************************************
*******************************************************************************
*****		Stream and related classes			*****
*******************************************************************************
*******************************************************************************
-->

<!--
*******************************************************************************
*****		Class: Stream			*****
*******************************************************************************
-->
<owl:Class rdf:ID="Stream">
</owl:Class>

<owl:DatatypeProperty rdf:ID="protocol">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="active">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="node1">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="node2">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="startTime">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="endTime">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="duration">
	<rdfs:domain rdf:resource="#Stream"/>
</owl:DatatypeProperty>


<!--
*****************************************************************************
*****		Class: L2Stream			*****
*****************************************************************************
-->
<owl:Class rdf:ID="L2Stream">
	<rdfs:subClassOf rdf:resource="#Stream"/>
</owl:Class>


<!--
*****************************************************************************
*****		Class: L3Stream			*****
*****************************************************************************
-->
<owl:Class rdf:ID="L3Stream">
	<rdfs:subClassOf rdf:resource="#L2Stream"/>
</owl:Class>



<!--
*****************************************************************************
*****		Class: L4Stream			*****
*****************************************************************************
-->
<owl:Class rdf:ID="L4Stream">
	<rdfs:subClassOf rdf:resource="#L3Stream"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="node1Port">
	<rdfs:domain rdf:resource="#L4Stream"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="node2Port">
	<rdfs:domain rdf:resource="#L4Stream"/>
</owl:DatatypeProperty>
```

```
<!--
*************************************************************************
*****              Class: TCPStream                    *****
*************************************************************************
-->
<owl:Class rdf:ID="TCPStream">
        <rdfs:comment xml:lang="en">
        A Stream that consists of two nodes sending TCP packets
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="#L4Stream"/>
        <owl:disjointWith rdf:resource="#UDPStream"/>
        <owl:disjointWith rdf:resource="#ICMPStream"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: UDPStream                    *****
*************************************************************************
-->
<owl:Class rdf:ID="UDPStream">
  <rdfs:comment xml:lang="en">
        A Stream that consists of two nodes sending UDP packets
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="#L4Stream"/>
        <owl:disjointWith rdf:resource="#TCPStream"/>
        <owl:disjointWith rdf:resource="#ICMPStream"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: ICMPStream                   *****
*************************************************************************
-->
<owl:Class rdf:ID="ICMPStream">
  <rdfs:comment xml:lang="en">
        A Stream that consists of two nodes sending ICMP packets
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="#L4Stream"/>
        <owl:disjointWith rdf:resource="#TCPStream"/>
        <owl:disjointWith rdf:resource="#UDPStream"/>
</owl:Class>
```

```
<!--
*******************************************************************************
*******************************************************************************
*****        PacketSequence and related classes              *****
*******************************************************************************
*******************************************************************************
-->

<!--
*******************************************************************************
*****              Class: PacketSequence                *****
*******************************************************************************
-->
<owl:Class rdf:ID="PacketSequence">
</owl:Class>

<owl:DatatypeProperty rdf:ID="seqID">
        <rdfs:domain rdf:resource="#PacketSequence"/>
</owl:DatatypeProperty>


<!--
*******************************************************************************
*****              Class: SeqItem                  *****
*******************************************************************************
-->
<owl:Class rdf:ID="SeqItem">
</owl:Class>

<owl:DatatypeProperty rdf:ID="seqParentID">
        <rdfs:domain rdf:resource="#SeqItem"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="orderNum">
        <rdfs:domain rdf:resource="#SeqItem"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="packet">
         <rdfs:domain rdf:resource="#SeqItem"/>
        <rdfs:range rdf:resource="#Packet"/>
</owl:ObjectProperty >
```

```
<!--
************************************************************************
************************************************************************
*****          Different Packet Types classes                *****
************************************************************************
************************************************************************
-->

<!--
************************************************************************
*****              Class: PingPacket                   *****
************************************************************************
-->
<owl:Class rdf:ID="PingPacket">
        <rdfs:comment>
          PingPacket are ICMPPackets with ICMPtype of 8 (echo request)
          One packet type for a possible Ping Flood attack
        </rdfs:comment>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#ICMPPacket"/>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#icmpType"/>
                        <owl:hasValue rdf:datatype="&xsd;integer">8</owl:hasValue>
                </owl:Restriction>
        </owl:intersectionOf>
</owl:Class>



<!--
************************************************************************
*****              Class: SmurfPacket                   *****
************************************************************************
-->
<owl:Class rdf:ID="SmurfPacket">
        <rdfs:comment>
          SmurfPacket are ICMPPackets with the last octet of destIP of 255
          One packet type for a possible Ping Flood attack
        </rdfs:comment>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#ICMPPacket"/>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#IPoctet4"/>
                        <owl:hasValue rdf:datatype="&xsd;integer">255</owl:hasValue>
                </owl:Restriction>
        </owl:intersectionOf>
</owl:Class>


<!--
*************************************************************************
*****              Class: SynPacket                    *****
*************************************************************************
-->
<owl:Class rdf:ID="SynPacket">
        <rdfs:comment>
          SynPacket are TCPPackets with the SYN flag set
          One packet type for a possible Port Scan attack
        </rdfs:comment>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#TCPPacket"/>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#tcpSynFlag"/>
                        <owl:hasValue rdf:datatype="&xsd;boolean">1</owl:hasValue>
                </owl:Restriction>
        </owl:intersectionOf>

</owl:Class>


<!--
*************************************************************************
*****              Class: FinPacket                    *****
*************************************************************************
-->
<owl:Class rdf:ID="FinPacket">
        <rdfs:comment>
          FinPacket are TCPPackets with FIN flag only set
          One packet type for a possible Port Scan attack
          Typically, TCP packets with FIN flag will also have ACK flag set
          TCP response to FIN flag only set will tell attacker if port is open
        </rdfs:comment>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#TCPPacket"/>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#tcpFinFlag"/>
                        <owl:hasValue rdf:datatype="&xsd;integer">1</owl:hasValue>
                </owl:Restriction>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#tcpAckFlag"/>
```

```
            <owl:hasValue rdf:datatype="&xsd;integer">0</owl:hasValue>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#tcpRstFlag"/>
            <owl:hasValue rdf:datatype="&xsd;integer">0</owl:hasValue>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#tcpSynFlag"/>
            <owl:hasValue rdf:datatype="&xsd;integer">0</owl:hasValue>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>


<!--
*************************************************************************
*****           Class: MaskPacket               *****
*************************************************************************
-->
<owl:Class rdf:ID="MaskPacket">
    <rdfs:comment>
      MaskPacket are ICMPPackets with ICMPtype of 17 (netmask request)
      One packet type for a possible ICMP Flood attack
    </rdfs:comment>
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#ICMPPacket"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#icmpType"/>
            <owl:hasValue rdf:datatype="&xsd;integer">17</owl:hasValue>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>



<!--
*************************************************************************
*************************************************************************
*****         Alert and related classes               *****
*************************************************************************
*************************************************************************
-->
```

```
<!--
*************************************************************************
*****                Class: Alert                    *****
*************************************************************************
-->
<owl:Class rdf:ID="Alert">
</owl:Class>

<owl:DatatypeProperty rdf:ID="aDateTime">
        <rdfs:domain rdf:resource="#Alert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aID">
        <rdfs:domain rdf:resource="#Alert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aDescription">
        <rdfs:domain rdf:resource="#Alert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aPriority">
        <rdfs:domain rdf:resource="#Alert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aClassification">
        <rdfs:domain rdf:resource="#Alert"/>
</owl:DatatypeProperty>

<!--
*************************************************************************
*****               Class: IPAlert                    *****
*************************************************************************
-->
<owl:Class rdf:ID="IPAlert">
        <rdfs:subClassOf rdf:resource="#Alert"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="aIPHdrLen">
        <rdfs:domain rdf:resource="#IPAlert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aIPDgramLen">
        <rdfs:domain rdf:resource="#IPAlert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aIPID">
        <rdfs:domain rdf:resource="#IPAlert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aIPProtocol">
        <rdfs:domain rdf:resource="#IPAlert"/>
</owl:DatatypeProperty>
```

```
<!--
*******************************************************************************
*****              Class: ICMPAlert                    *****
*******************************************************************************
-->
<owl:Class rdf:ID="ICMPAlert">
        <rdfs:subClassOf rdf:resource="#IPAlert"/>
        <owl:disjointWith rdf:resource="#L4Alert"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="aICMPType">
         <rdfs:domain rdf:resource="#ICMPAlert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="aICMPCode">
        <rdfs:domain rdf:resource="#ICMPAlert"/>
</owl:DatatypeProperty>


<!--
*******************************************************************************
*****              Class: L4Alert                    *****
*******************************************************************************
-->
<owl:Class rdf:ID="L4Alert">
        <rdfs:subClassOf rdf:resource="#IPAlert"/>
        <owl:disjointWith rdf:resource="#ICMPAlert"/>
</owl:Class>

<owl:DatatypeProperty rdf:about="#aL4SrcPort">
        <rdfs:domain rdf:resource="#L4Alert"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#aL4DestPort">
        <rdfs:domain rdf:resource="#L4Alert"/>
</owl:DatatypeProperty>


<!--
*******************************************************************************
*****              Class: TCPAlert                    *****
*******************************************************************************
-->
<owl:Class rdf:ID="TCPAlert">
        <rdfs:subClassOf rdf:resource="#L4Alert"/>
```

```
          <owl:disjointWith rdf:resource="#UDPAlert"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="aTCPFlags">
          <rdfs:domain rdf:resource="#TCPAlert"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="aTCPSeqNum">
          <rdfs:domain rdf:resource="#TCPAlert"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="aTCPAckNum">
          <rdfs:domain rdf:resource="#TCPAlert"/>
  </owl:DatatypeProperty>

  <!--
  ************************************************************************
  *****                 Class: UDPAlert                      *****
  ************************************************************************
  -->
  <owl:Class rdf:ID="UDPAlert">
          <rdfs:subClassOf rdf:resource="#L4Alert"/>
          <owl:disjointWith rdf:resource="#TCPAlert"/>
  </owl:Class>


</rdf:RDF>
```

B.1 Attack Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY attack
'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl#'>
   <!ENTITY traffic
'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl#'>
   <!ENTITY complex
'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/complexAttack.owl#'>
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
]>

<!--
 *******************************************************************************
 *******************************************************************************
 *****                    Attack Ontology                          *****
 *******************************************************************************
 *******************************************************************************
-->

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:attack="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl#"
   xmlns:traffic="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl#"
   xmlns:complex=
         "http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/complexAttack.owl#"
   xml:base="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl">

 <owl:Ontology rdf:about="">
     <rdfs:comment>An ontology for network attacks</rdfs:comment>
 </owl:Ontology>



 <!--
 *******************************************************************************
 *******************************************************************************
 *****                Object Property Definitions                      *****
 *******************************************************************************
 *******************************************************************************
 -->
```

```
<!--
*****        Object Properties: Attacks for IP address          *****
-->
<owl:ObjectProperty rdf:ID="wasAttacked">
    <rdfs:range rdf:resource="#Attack"/>
    <owl:inverseOf rdf:resource="#hasTargetIP"/>
</owl:ObjectProperty>



<!--
*************************************************************************
*************************************************************************
*****                AttackPacket Class                      *****
*************************************************************************
*************************************************************************
-->
<owl:Class rdf:ID="AttackPacket">
</owl:Class>
<owl:ObjectProperty rdf:ID="hasOrigMAC">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#MACaddress"/>
    <rdfs:domain rdf:resource="#AttackPacket"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasTargetMAC">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#MACaddress"/>
    <rdfs:domain rdf:resource="#AttackPacket"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasOrigIP">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#IPaddress"/>
    <rdfs:domain rdf:resource="#AttackPacket"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasTargetIP">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#IPaddress"/>
    <rdfs:domain rdf:resource="#AttackPacket"/>
</owl:ObjectProperty>


<owl:DatatypeProperty rdf:ID="beginDate">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#AttackPacket"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="endDate">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#AttackPacket"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="pcType">
    <rdfs:domain rdf:resource="#PacketCollection"/>
    <rdfs:range rdf:resource="&traffic;PacketType"/>
</owl:ObjectProperty>

<!-- Restriction on type property -->
<owl:Class rdf:about="#PacketCollection">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#pcType"/>
            <owl:allValuesFrom rdf:resource="&traffic;PacketType"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!--
*************************************************************************
*************************************************************************
*****              PacketCollection Class                    *****
*************************************************************************
*************************************************************************
-->

<owl:Class rdf:ID="PacketCollection">
</owl:Class>

<owl:DatatypeProperty rdf:ID="pcFrequency">
</owl:DatatypeProperty>
```

```
<!--
 *******************************************************************************
 *******************************************************************************
 *****                SimpleAttack Class                    *****
 *******************************************************************************
 *******************************************************************************
 -->

 <owl:Class rdf:ID="SimpleAttack">
     <owl:unionOf rdf:parseType="Collection">
           <owl:Class rdf:about="#Availability"/>
           <owl:Class rdf:about="#Recon"/>
           <owl:Class rdf:about="#GainAccess"/>
           <owl:Class rdf:about="#ViewChangeData"/>
     </owl:unionOf>
 </owl:Class>


 <!--
 *******************************************************************************
 *******************************************************************************
 *****                  Attack Class                    *****
 *******************************************************************************
 *******************************************************************************
 -->
 <owl:Class rdf:ID="Attack">
     <rdfs:subClassOf rdf:resource="#AttackPacket"/>
 </owl:Class>

 <owl:DatatypeProperty rdf:ID="name">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="description">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="preconds">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="postconds">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="priority">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="consequence">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="motivation">
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="attTimeInt">
 </owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="remedy">
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="snortPriority">
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="attPacketSeq">
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="attStream">
</owl:DatatypeProperty>



<!--
*********************************************************************************
*********************************************************************************
*****              Availability and Related Classes              *****
*********************************************************************************
*********************************************************************************
-->

<!--
*********************************************************************************
*****               Class: Availability                          *****
*********************************************************************************
-->
<owl:Class rdf:ID="Availability">
    <rdfs:subClassOf rdf:resource="#Attack"/>
    <owl:disjointWith rdf:resource="#Recon"/>
    <owl:disjointWith rdf:resource="#GainAccess"/>
    <owl:disjointWith rdf:resource="#ViewChangeData"/>
</owl:Class>


<!--
*********************************************************************************
*****               Class: DoS                               *****
*********************************************************************************
-->
<owl:Class rdf:ID="DoS">
    <rdfs:subClassOf rdf:resource="#Availability"/>
    <owl:disjointWith rdf:resource="#Spoofing"/>
</owl:Class>
```

```
<!--
********************************************************************************
*****              Class: Resources                    *****
********************************************************************************
-->
<owl:Class rdf:ID="Resources">
   <rdfs:subClassOf rdf:resource="#DoS"/>
   <owl:disjointWith rdf:resource="#CrashNode"/>
</owl:Class>


<!--
********************************************************************************
*****              Class: Flood                        *****
********************************************************************************
-->
<owl:Class rdf:ID="Flood">
   <rdfs:subClassOf rdf:resource="#Resources"/>
   <owl:disjointWith rdf:resource="#Memory"/>
   <owl:disjointWith rdf:resource="#CPU"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="floodFrequency">
   <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
   <owl:equivalentProperty rdf:resource="#pcFrequency"/>
</owl:DatatypeProperty>


<!--
********************************************************************************
*****              Class: PingFlood                    *****
********************************************************************************
-->
<owl:Class rdf:ID="PingFlood">
   <rdfs:comment>
     A PingFlood packet is an instance of the PacketCollection
     of type PingFloodType with greater than "threshold" frequency.
   </rdfs:comment>

   <rdfs:subClassOf rdf:resource="#Flood"/>
   <owl:disjointWith rdf:resource="#ICMPFlood"/>
   <owl:disjointWith rdf:resource="#TCPFlood"/>
   <owl:disjointWith rdf:resource="#AppFlood"/>
```

```
  <owl:intersectionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#PacketCollection"/>
         <owl:Restriction>
              <owl:onProperty rdf:resource="#pcType"/>
              <owl:hasValue rdf:resource="&traffic;PingFloodType"/>
         </owl:Restriction>
  </owl:intersectionOf>

</owl:Class>



<!--
*************************************************************************
*****              Class: ICMPFlood                         *****
*************************************************************************
-->
<owl:Class rdf:ID="ICMPFlood">
  <rdfs:comment>
        A ICMPFlood packet is an instance of the PacketCollection
        of type ICMPFloodType with greater than "threshold" frequency.
     </rdfs:comment>

     <rdfs:subClassOf rdf:resource="#Flood"/>
     <owl:disjointWith rdf:resource="#PingFlood"/>
     <owl:disjointWith rdf:resource="#TCPFlood"/>
     <owl:disjointWith rdf:resource="#AppFlood"/>

  <owl:intersectionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#PacketCollection"/>
         <owl:Restriction>
              <owl:onProperty rdf:resource="#pcType"/>
              <owl:hasValue rdf:resource="&traffic;ICMPFloodType"/>
         </owl:Restriction>
     </owl:intersectionOf>

</owl:Class>



<!--
*************************************************************************
*****              Class: TCPFlood                         *****
*************************************************************************
-->
```

```
<owl:Class rdf:ID="TCPFlood">
  <rdfs:comment>
       A TCPFlood packet is an instance of the PacketCollection
       of type TCPType with greater than "threshold" frequency.
     </rdfs:comment>

     <rdfs:subClassOf rdf:resource="#Flood"/>
     <owl:disjointWith rdf:resource="#PingFlood"/>
     <owl:disjointWith rdf:resource="#ICMPFlood"/>
     <owl:disjointWith rdf:resource="#AppFlood"/>

  <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#PacketCollection"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#pcType"/>
            <owl:hasValue rdf:resource="&traffic;TCPFloodType"/>
        </owl:Restriction>
  </owl:intersectionOf>

</owl:Class>


<!--
************************************************************************
*****              Class: AppFlood                         *****
************************************************************************
-->
<owl:Class rdf:ID="AppFlood">
  <rdfs:comment>
       A AppFlood packet is an instance of the PacketCollection
       of type AppFloodType with greater than "threshold" frequency.
     </rdfs:comment>

     <rdfs:subClassOf rdf:resource="#Flood"/>
     <owl:disjointWith rdf:resource="#PingFlood"/>
     <owl:disjointWith rdf:resource="#ICMPFlood"/>
     <owl:disjointWith rdf:resource="#TCPFlood"/>

  <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#PacketCollection"/>
      <owl:Restriction>
          <owl:onProperty rdf:resource="#pcType"/>
          <owl:hasValue rdf:resource="&traffic;AppFloodType"/>
      </owl:Restriction>
  </owl:intersectionOf>
```

```
</owl:Class>


<!--
*************************************************************************
*****            Class: Memory                        *****
*************************************************************************
-->
<owl:Class rdf:ID="Memory">
    <rdfs:subClassOf rdf:resource="#Resources"/>
    <owl:disjointWith rdf:resource="#Flood"/>
    <owl:disjointWith rdf:resource="#CPU"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="memAvail">
    <rdfs:domain rdf:resource="#Memory"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="memUsed">
    <rdfs:domain rdf:resource="#Memory"/>
</owl:DatatypeProperty>


<!--
*************************************************************************
*****            Class: CPU                        *****
*************************************************************************
-->
<owl:Class rdf:ID="CPU">
    <rdfs:subClassOf rdf:resource="#Resources"/>
    <owl:disjointWith rdf:resource="#Flood"/>
    <owl:disjointWith rdf:resource="#Memory"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="cpuAmount">
    <rdfs:domain rdf:resource="#CPU"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="cpuPercUsed">
    <rdfs:domain rdf:resource="#CPU"/>
</owl:DatatypeProperty>
```

```
<!--
*************************************************************************
*****              Class: CrashNode                         *****
*************************************************************************
-->
<owl:Class rdf:ID="CrashNode">
    <rdfs:subClassOf rdf:resource="#DoS"/>
    <owl:disjointWith rdf:resource="#Resources"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: Land                         *****
*************************************************************************
-->
<owl:Class rdf:ID="Land">
  <rdfs:comment>
      A Land packet is a TCPPacket with DIP = SIP and DPort = Sport
    </rdfs:comment>

    <rdfs:subClassOf rdf:resource="#CrashNode"/>
    <owl:disjointWith rdf:resource="#Teardrop"/>
    <owl:disjointWith rdf:resource="#PingOfDeath"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: Teardrop                         *****
*************************************************************************
-->
<owl:Class rdf:ID="Teardrop">
  <rdfs:comment>
      A Teardrop packet is a PacketSequence with multiple packets
      with same SIP and overlapping, oversized payloads
    </rdfs:comment>

    <rdfs:subClassOf rdf:resource="#CrashNode"/>
    <owl:disjointWith rdf:resource="#Land"/>
    <owl:disjointWith rdf:resource="#PingOfDeath"/>

    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#PacketCollection"/>
      <owl:Restriction>
```

```
          <owl:onProperty rdf:resource="#pcType"/>
          <owl:hasValue rdf:resource="&traffic;TeardropType"/>
      </owl:Restriction>
  </owl:intersectionOf>

</owl:Class>


<!--
************************************************************************
*****               Class: PingOfDeath                    *****
************************************************************************
-->
<owl:Class rdf:ID="PingOfDeath">
    <rdfs:comment>
       PoDPacket (Ping of Death) are ICMPPackets with ICMPtype of 8
       (echo request) and packetLen of 65535 (should really be -ge 65535)
       One packet type for a possible Ping Flood attack causing buffer overflow
    </rdfs:comment>

    <rdfs:subClassOf rdf:resource="#CrashNode"/>
    <owl:disjointWith rdf:resource="#Land"/>
    <owl:disjointWith rdf:resource="#Teardrop"/>

  <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&traffic;ICMPPacket"/>
      <owl:Restriction>
          <owl:onProperty rdf:resource="&traffic;icmpType"/>
          <owl:hasValue rdf:datatype="&xsd;integer">8</owl:hasValue>
      </owl:Restriction>
      <owl:Restriction>
          <owl:onProperty rdf:resource="&traffic;packetLen"/>
          <owl:hasValue rdf:datatype="&xsd;integer">65535</owl:hasValue>
      </owl:Restriction>
  </owl:intersectionOf>

</owl:Class>


<!--
************************************************************************
*****               Class: Spoofing                    *****
************************************************************************
-->
<owl:Class rdf:ID="Spoofing">
```

```
    <rdfs:subClassOf rdf:resource="#Availability"/>
    <owl:disjointWith rdf:resource="#DoS"/>
</owl:Class>


<!--
************************************************************************
*****              Class: ARPSpoof                    *****
************************************************************************
-->
<owl:Class rdf:ID="ARPSpoof">
    <rdfs:subClassOf rdf:resource="#Spoofing"/>
    <owl:disjointWith rdf:resource="#IPSpoof"/>
</owl:Class>


<!--
************************************************************************
*****              Class: IPSpoof                     *****
************************************************************************
-->
<owl:Class rdf:ID="IPSpoof">
    <rdfs:subClassOf rdf:resource="#Spoofing"/>
    <owl:disjointWith rdf:resource="#ARPSpoof"/>
</owl:Class>


<!--
************************************************************************
************************************************************************
*****               Recon and Related Classes              *****
************************************************************************
************************************************************************
-->

<!--
************************************************************************
*****              Class: Recon                       *****
************************************************************************
-->
<owl:Class rdf:ID="Recon">
    <rdfs:subClassOf rdf:resource="#Attack"/>
    <owl:disjointWith rdf:resource="#Availability"/>
    <owl:disjointWith rdf:resource="#GainAccess"/>
    <owl:disjointWith rdf:resource="#ViewChangeData"/>
```

```
</owl:Class>

<owl:DatatypeProperty rdf:ID="reconPortNum">
</owl:DatatypeProperty>


<!--
**********************************************************************
*****              Class: Scan                         *****
**********************************************************************
-->
<owl:Class rdf:ID="Scan">
   <rdfs:subClassOf rdf:resource="#Recon"/>
   <owl:disjointWith rdf:resource="#GatherInfo"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="scanFrequency">
   <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
   <owl:equivalentProperty rdf:resource="#pcFrequency"/>
</owl:DatatypeProperty>



<!--
**********************************************************************
*****              Class: PingScan                     *****
**********************************************************************
-->
<owl:Class rdf:ID="PingScan">
   <rdfs:comment>
     A PingScan packet is an instance of the PacketCollection
     of type PingScanType.
   </rdfs:comment>

   <rdfs:subClassOf rdf:resource="#Scan"/>
   <owl:disjointWith rdf:resource="#NodeScan"/>

 <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#PacketCollection"/>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#pcType"/>
         <owl:hasValue rdf:resource="&traffic;PingScanType"/>
      </owl:Restriction>
 </owl:intersectionOf>
```

```
</owl:Class>


<!--
*************************************************************************
*****            Class: NodeScan                    *****
*************************************************************************
-->
<owl:Class rdf:ID="NodeScan">
    <rdfs:subClassOf rdf:resource="#Scan"/>
    <owl:disjointWith rdf:resource="#PingScan"/>
</owl:Class>


<!--
*************************************************************************
*****            Class: PortScan                    *****
*************************************************************************
-->
<owl:Class rdf:ID="PortScan">
    <rdfs:subClassOf rdf:resource="#NodeScan"/>
    <owl:disjointWith rdf:resource="#SYNScan"/>
    <owl:disjointWith rdf:resource="#FINScan"/>
    <owl:disjointWith rdf:resource="#NULLScan"/>
    <owl:disjointWith rdf:resource="#TCPConnect"/>

  <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#PacketCollection"/>
      <owl:Restriction>
          <owl:onProperty rdf:resource="#pcType"/>
          <owl:hasValue rdf:resource="&traffic;PortScanType"/>
      </owl:Restriction>
  </owl:intersectionOf>

</owl:Class>


<!--
*************************************************************************
*****            Class: SYNScan                     *****
*************************************************************************
-->
<owl:Class rdf:ID="SYNScan">
    <rdfs:subClassOf rdf:resource="#NodeScan"/>
    <owl:disjointWith rdf:resource="#PortScan"/>
    <owl:disjointWith rdf:resource="#FINScan"/>
    <owl:disjointWith rdf:resource="#NULLScan"/>
    <owl:disjointWith rdf:resource="#TCPConnect"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: FINScan                        *****
*************************************************************************
-->
<owl:Class rdf:ID="FINScan">
    <rdfs:subClassOf rdf:resource="#NodeScan"/>
    <owl:disjointWith rdf:resource="#PortScan"/>
    <owl:disjointWith rdf:resource="#SYNScan"/>
    <owl:disjointWith rdf:resource="#NULLScan"/>
    <owl:disjointWith rdf:resource="#TCPConnect"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: NULLScan                       *****
*************************************************************************
-->
<owl:Class rdf:ID="NULLScan">
    <rdfs:comment>
      NullPacket are TCPPackets with no flags set
      One packet type for a possible Port Scan attack
    </rdfs:comment>

    <rdfs:subClassOf rdf:resource="#NodeScan"/>
    <owl:disjointWith rdf:resource="#PortScan"/>
    <owl:disjointWith rdf:resource="#SYNScan"/>
    <owl:disjointWith rdf:resource="#FINScan"/>
    <owl:disjointWith rdf:resource="#TCPConnect"/>

  <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="&traffic;TCPPacket"/>
      <owl:Restriction>
          <owl:onProperty rdf:resource="&traffic;tcpFlags"/>
          <owl:hasValue rdf:datatype="&xsd;integer">0</owl:hasValue>
      </owl:Restriction>
  </owl:intersectionOf>
```

```
</owl:Class>



<!--
*************************************************************************
*****              Class: TCPConnect                    *****
*************************************************************************
-->
<owl:Class rdf:ID="TCPConnect">
   <rdfs:comment>
       Mitnick sent SYN request to X-Terminal and received SYN/ACK response.
       Then he sent RESET response to keep the X-Terminal from being filled up.
       For our purposes, we will look for multiple TCPPackets to the same
       destination IP address with the RST flag set.
   </rdfs:comment>

   <rdfs:subClassOf rdf:resource="#NodeScan"/>
   <owl:disjointWith rdf:resource="#PortScan"/>
   <owl:disjointWith rdf:resource="#SYNScan"/>
   <owl:disjointWith rdf:resource="#FINScan"/>
   <owl:disjointWith rdf:resource="#NULLScan"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: GatherInfo                   *****
*************************************************************************
-->
<owl:Class rdf:ID="GatherInfo">
   <rdfs:subClassOf rdf:resource="#Recon"/>
   <owl:disjointWith rdf:resource="#Scan"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="infoLearned">
</owl:DatatypeProperty>



<!--
*************************************************************************
*****              Class: Sniffing                   *****
*************************************************************************
-->
```

```
<owl:Class rdf:ID="Sniffing">
    <rdfs:subClassOf rdf:resource="#GatherInfo"/>
    <owl:disjointWith rdf:resource="#InfoLeak"/>
</owl:Class>


<!--
************************************************************************
*****              Class: NodeInfo                         *****
************************************************************************
-->
<owl:Class rdf:ID="NodeInfo">
    <rdfs:subClassOf rdf:resource="#Sniffing"/>
    <owl:disjointWith rdf:resource="#UserInfo"/>
    <owl:disjointWith rdf:resource="#TCPInfo"/>
</owl:Class>


<!--
************************************************************************
*****              Class: UserInfo                         *****
************************************************************************
-->
<owl:Class rdf:ID="UserInfo">
    <rdfs:subClassOf rdf:resource="#Sniffing"/>
    <owl:disjointWith rdf:resource="#NodeInfo"/>
    <owl:disjointWith rdf:resource="#TCPInfo"/>
</owl:Class>


<!--
************************************************************************
*****              Class: TCPInfo                          *****
************************************************************************
-->
<owl:Class rdf:ID="TCPInfo">
    <rdfs:subClassOf rdf:resource="#Sniffing"/>
    <owl:disjointWith rdf:resource="#NodeInfo"/>
    <owl:disjointWith rdf:resource="#UserInfo"/>
</owl:Class>
```

```
<!--
*************************************************************************
*****              Class: InfoLeak                         *****
*************************************************************************
-->
<owl:Class rdf:ID="InfoLeak">
    <rdfs:subClassOf rdf:resource="#GatherInfo"/>
    <owl:disjointWith rdf:resource="#Sniffing"/>
</owl:Class>


<!--
*************************************************************************
*************************************************************************
*****             GainAccess and Related Classes              *****
*************************************************************************
*************************************************************************
-->

<!--
*************************************************************************
*****              Class: GainAccess                         *****
*************************************************************************
-->
<owl:Class rdf:ID="GainAccess">
    <rdfs:subClassOf rdf:resource="#Attack"/>
    <owl:disjointWith rdf:resource="#Availability"/>
    <owl:disjointWith rdf:resource="#Recon"/>
    <owl:disjointWith rdf:resource="#ViewChangeData"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: UnauthAccess                         *****
*************************************************************************
-->
<owl:Class rdf:ID="UnauthAccess">
    <rdfs:subClassOf rdf:resource="#GainAccess"/>
    <owl:disjointWith rdf:resource="#PrivilegeGain"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="uaPortNum">
</owl:DatatypeProperty>
```

```
<!--
*************************************************************************
*****           Class: PrivilegeGain               *****
*************************************************************************
-->
<owl:Class rdf:ID="PrivilegeGain">
   <rdfs:subClassOf rdf:resource="#GainAccess"/>
   <owl:disjointWith rdf:resource="#UnauthAccess"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="pgValue">
</owl:DatatypeProperty>


<!--
*************************************************************************
*****           Class: User                        *****
*************************************************************************
-->
<owl:Class rdf:ID="UserPG">
   <rdfs:subClassOf rdf:resource="#PrivilegeGain"/>
   <owl:disjointWith rdf:resource="#AdminPG"/>
   <owl:disjointWith rdf:resource="#RootPG"/>
</owl:Class>


<!--
*************************************************************************
*****           Class: Admin                       *****
*************************************************************************
-->
<owl:Class rdf:ID="AdminPG">
   <rdfs:subClassOf rdf:resource="#PrivilegeGain"/>
   <owl:disjointWith rdf:resource="#UserPG"/>
   <owl:disjointWith rdf:resource="#RootPG"/>
</owl:Class>


<!--
*************************************************************************
*****           Class: Root                        *****
*************************************************************************
-->
```

```
<owl:Class rdf:ID="RootPG">
    <rdfs:subClassOf rdf:resource="#PrivilegeGain"/>
    <owl:disjointWith rdf:resource="#UserPG"/>
    <owl:disjointWith rdf:resource="#AdminPG"/>
</owl:Class>




<!--
*************************************************************************
*************************************************************************
*****            ViewChangeData and Related Classes            *****
*************************************************************************
*************************************************************************
-->

<!--
*************************************************************************
*****              Class: ViewChangeData                     *****
*************************************************************************
-->
<owl:Class rdf:ID="ViewChangeData">
    <rdfs:subClassOf rdf:resource="#Attack"/>
    <owl:disjointWith rdf:resource="#Availability"/>
    <owl:disjointWith rdf:resource="#Recon"/>
    <owl:disjointWith rdf:resource="#GainAccess"/>
</owl:Class>


<!--
*************************************************************************
*****              Class: MaliciousCode                     *****
*************************************************************************
-->
<owl:Class rdf:ID="MaliciousCode">
    <rdfs:subClassOf rdf:resource="#ViewChangeData"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="mcService">
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="mcPortNum">
</owl:DatatypeProperty>
```

```
<!--
************************************************************************
*****              Class: RPC                         *****
************************************************************************
-->
<owl:Class rdf:ID="RPC">
    <rdfs:subClassOf rdf:resource="#MaliciousCode"/>
    <owl:disjointWith rdf:resource="#ExecCode"/>
    <owl:disjointWith rdf:resource="#WebServer"/>
    <owl:disjointWith rdf:resource="#SendFile"/>
    <owl:disjointWith rdf:resource="#SystemCall"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="rpcCode">
</owl:DatatypeProperty>


<!--
************************************************************************
*****              Class: ExecCode                      *****
************************************************************************
-->
<owl:Class rdf:ID="ExecCode">
    <rdfs:subClassOf rdf:resource="#MaliciousCode"/>
    <owl:disjointWith rdf:resource="#RPC"/>
    <owl:disjointWith rdf:resource="#WebServer"/>
    <owl:disjointWith rdf:resource="#SendFile"/>
    <owl:disjointWith rdf:resource="#SystemCall"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="ecCode">
</owl:DatatypeProperty>


<!--
************************************************************************
*****              Class: WebServer                     *****
************************************************************************
-->
<owl:Class rdf:ID="WebServer">
    <rdfs:subClassOf rdf:resource="#MaliciousCode"/>
    <owl:disjointWith rdf:resource="#RPC"/>
    <owl:disjointWith rdf:resource="#ExecCode"/>
    <owl:disjointWith rdf:resource="#SendFile"/>
    <owl:disjointWith rdf:resource="#SystemCall"/>
```

```
  </owl:Class>


  <!--
  *************************************************************************
  *****           Class: SendFile                 *****
  *************************************************************************
  -->
  <owl:Class rdf:ID="SendFile">
     <rdfs:subClassOf rdf:resource="#MaliciousCode"/>
     <owl:disjointWith rdf:resource="#RPC"/>
     <owl:disjointWith rdf:resource="#ExecCode"/>
     <owl:disjointWith rdf:resource="#WebServer"/>
     <owl:disjointWith rdf:resource="#SystemCall"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="filename">
  </owl:DatatypeProperty>


  <!--
  *************************************************************************
  *****           Class: SystemCall                 *****
  *************************************************************************
  -->
  <owl:Class rdf:ID="SystemCall">
     <rdfs:subClassOf rdf:resource="#MaliciousCode"/>
     <owl:disjointWith rdf:resource="#RPC"/>
     <owl:disjointWith rdf:resource="#ExecCode"/>
     <owl:disjointWith rdf:resource="#WebServer"/>
     <owl:disjointWith rdf:resource="#SendFile"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="sysCall">
  </owl:DatatypeProperty>


</rdf:RDF>
```

B.3 Complex Attack Ontology Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY complex
         'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/complexAttack.owl#'>
   <!ENTITY attack
         'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl#'>
   <!ENTITY traffic
         'http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl#'>
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
   <!ENTITY time 'http://www.w3.org/TR/owl-time#'>
]>

<!--
 ************************************************************************
 ************************************************************************
 *****         Complex Attack Ontology           *****
 ************************************************************************
 ************************************************************************
-->


<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:time="http://www.w3.org/TR/owl-time#"
   xmlns:complex=
         "http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/complexAttack.owl#"
   xml:base=
         "http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/complexAttack.owl">

  <owl:Ontology rdf:about="">
         <rdfs:comment>An ontology for complex attacks</rdfs:comment>
   <owl:imports
    rdf:resource="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/attack.owl"/>
   <owl:imports
    rdf:resource="http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/traffic.owl"/>
  </owl:Ontology>
```

```
<!--
************************************************************************
************************************************************************
*****              Complex Attack Classes                    *****
************************************************************************
************************************************************************
-->

<owl:Class rdf:ID="ComplexAttack">
  <rdfs:comment>
        A complex attack
  </rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="caHasTargetIP">
        <rdfs:domain rdf:resource="#ComplexAttack"/>
        <owl:equivalentProperty rdf:resource="&attack;hasTargetIP"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="caBeginDate">
        <rdfs:domain rdf:resource="#ComplexAttack"/>
        <owl:equivalentProperty rdf:resource="&attack;beginDate"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="caEndDate">
        <rdfs:domain rdf:resource="#ComplexAttack"/>
        <owl:equivalentProperty rdf:resource="&attack;endDate"/>
</owl:DatatypeProperty>




<!--
************************************************************************
*****              Class: DoSComplex                         *****
************************************************************************
-->
<owl:Class rdf:ID="DoSComplex">
        <rdfs:comment>
          A complex DoS attack is a Ping scan, Node scan, and Availability attack
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="#ComplexAttack"/>

        <owl:equivalentClass>
        <owl:Class>
                  <owl:intersectionOf rdf:parseType="Collection">
                           <owl:Class rdf:about="&traffic;NWaddressScanned"/>
```

```
            <rdf:Description rdf:about="&traffic;IPaddress"/>
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="&attack;wasAttacked"/>
                <owl:someValuesFrom
                        rdf:resource="&attack;Availability"/>
            </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>

</owl:Class>



<!--
*************************************************************************
*****           Class: PrivilegeEscalation              *****
*************************************************************************
-->
<owl:Class rdf:ID="PrivilegeEscalation">
    <rdfs:comment>
      A complex Privilege Escalation attack is a GainAccess instance OR the
      combination of a Ping scan, Node scan, and Gather Information attack.
    </rdfs:comment>

    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&attack;GainAccess"/>
        <owl:Class>
            <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Class
                        rdf:about="&traffic;NWaddressScanned"/>
                    <rdf:Description rdf:about="&traffic;IPaddress"/>
                    <owl:Restriction>
                      <owl:onProperty
                        rdf:resource="&attack;wasAttacked"/>
                      <owl:someValuesFrom
                        rdf:resource="&attack;NodeScan"/>
                    </owl:Restriction>
                        <owl:Restriction>
                          <owl:onProperty
                           rdf:resource="&attack;wasAttacked"/>
                          <owl:someValuesFrom
                            rdf:resource="&attack;GatherInfo"/>
                    </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:equivalentClass>
        </owl:Class>
    </owl:unionOf>

</owl:Class>


<!--
*************************************************************************
*****           Class: ConfIntLoss                    *****
*************************************************************************
-->
<owl:Class rdf:ID="ConfIntLoss">
    <rdfs:comment>
      A complex Confidentiality or Integrity Loss attack is a Ping scan,
      Node scan, and Malicious Code attack
    </rdfs:comment>

    <owl:equivalentClass>
    <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="&traffic;NWaddressScanned"/>
            <rdf:Description rdf:about="&traffic;IPaddress"/>
            <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="&attack;wasAttacked"/>
                <owl:someValuesFrom
                        rdf:resource="&attack;NodeScan"/>
            </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="&attack;wasAttacked"/>
                <owl:someValuesFrom
                        rdf:resource="&attack;MaliciousCode"/>
            </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>

</owl:Class>
```

```
<!--
*************************************************************************
*****             Class: Hijacking                     *****
*************************************************************************
-->
<owl:Class rdf:ID="Hijacking">
     <rdfs:comment>
       A complex Hijacking attack is a Ping scan, Node scan, TCP Scan against
       one host (host A) and an Availability and Spoofing attack against another
       host (host B) that has a current TCP connection with the first host (host A)
     </rdfs:comment>

 <owl:equivalentClass>
     <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
           <owl:Class rdf:about="&traffic;NWaddressScanned"/>
           <rdf:Description rdf:about="&traffic;IPaddress"/>
           <owl:Restriction>
                 <owl:onProperty rdf:resource="&attack;wasAttacked"/>
                 <owl:someValuesFrom rdf:resource="&attack;NodeScan"/>
           </owl:Restriction>
           <owl:Restriction>
                 <owl:onProperty rdf:resource="&attack;wasAttacked"/>
                 <owl:someValuesFrom rdf:resource="&attack;TCPConnect"/>
           </owl:Restriction>
           <owl:Restriction>
                 <!-- host A has a TCP connection (stream) with a host B that has
                     had an availability and spoof  attack against it   -->
                 <owl:onProperty rdf:resource="&traffic;hasTCPStreamWith"/>
                 <owl:someValuesFrom>
                       <owl:Class>
                          <owl:intersectionOf rdf:parseType="Collection">
                             <rdf:Description rdf:about="&traffic;IPaddress"/>
                             <owl:Restriction>
                                   <owl:onProperty
                                      rdf:resource="&attack;wasAttacked"/>
                                   <owl:someValuesFrom
                                      rdf:resource="&attack;Availability"/>
                             </owl:Restriction>
                             <owl:Restriction>
                                   <owl:onProperty
                                      rdf:resource="&attack;wasAttacked"/>
                                   <owl:someValuesFrom
                                      rdf:resource="&attack;Spoofing"/>
                             </owl:Restriction>
```

```
                      </owl:intersectionOf>
                    </owl:Class>
               </owl:someValuesFrom>
          </owl:Restriction>
       </owl:intersectionOf>
     </owl:Class>
  </owl:equivalentClass>

 </owl:Class>


</rdf:RDF>
```

Appendix C   SPARQL Rules in TRIDSO

This appendix contains the Java files for the prototype system (TRIDSO) that contain SPARQL rules. These rules are used to add instances to the knowledge base for attack detection. Only the Java files containing SPARQL rules are included in the appendix; files not containing SPARQL rules are not included. All of the source code for TRIDSO can be downloaded at http://faculty.kutztown.edu/frye/res/index.html.

C.1 Java File to Create Packet Collection Instances

```
/************************************************************************/
/*                                                          */
/* Author: Lisa Frye                                        */
/* Date: February 2011                                      */
/* Filename: PacketCollections.java                         */
/*                                                          */
/* Description: This file contains functions to execute SPARQL     */
/*              queries against the KB and add instances for       */
/*              packet collections.                         */
/* API: this program uses the Jena ontology API.              */
/*                                                          */
/************************************************************************/


// imports for Jena API
import com.hp.hpl.jena.update.GraphStore;
import com.hp.hpl.jena.update.GraphStoreFactory;
import com.hp.hpl.jena.update.UpdateAction;
import com.hp.hpl.jena.update.UpdateFactory;
import com.hp.hpl.jena.update.UpdateProcessor;
import com.hp.hpl.jena.update.UpdateRequest;
import com.hp.hpl.jena.update.UpdateExecutionFactory;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.Syntax;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.datatypes.xsd.XSDDatatype;
import com.hp.hpl.jena.datatypes.xsd.XSDDateTime;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.DatatypeProperty;
import com.hp.hpl.jena.ontology.ObjectProperty;
import com.hp.hpl.jena.datatypes.xsd.XSDDatatype;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;

// general imports
import java.lang.*;
import java.io.*;
import java.util.*;
import java.util.Iterator;
import java.util.Collection;
import java.util.ArrayList;
import java.text.DecimalFormat;



public class PacketCollections {


  // variables to time adding instances via SPARQL
     private static double sparqlTime = 0;
     private static String sparqlTimeSt;

     private static DecimalFormat decVal = new DecimalFormat ("#0.0000000");

     public static final String URL_PREFIX =
              "http://faculty.kutztown.edu/frye/res/onto/reason/tridso_v1/";

     private static final String TRAFFICONT = KButility.URL_PREFIX + "traffic";
     private static final String TRAFFICONT_URL = TRAFFICONT + ".owl";
     private static final String TRAFFICONT_PREFIX = TRAFFICONT_URL + "#";
     private static final String ATTACKONT = KButility.URL_PREFIX + "attack";
     private static final String ATTACKONT_URL = ATTACKONT + ".owl";
     private static final String ATTACKONT_PREFIX = ATTACKONT_URL + "#";
```

```
/*****************************************************************/
/*****                                           *****/
/*****        Add PingFlood PacketColletion Instances             *****/
/*****                                           *****/
/*****************************************************************/
public static double addPingFloods(PrintStream outputFile,
                        double addCollectionsTime) {

try {

     System.out.println("\tAdding Ping Flood instances...");
    outputFile.println("Adding PingFloodType instances from PingPacket...");

   // Retrieve all PingPacket instances to same destIPs
   String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX attack: " +
   "<" + KButility.URL_PREFIX + "attack.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "INSERT " +
    "{" +
   "   _:a rdf:type attack:PacketCollection; " +
   "      attack:beginDate ?beginDateTime; " +
   "      attack:endDate ?endDateTime; " +
    "      attack:pcType traffic:PingFloodType; " +
    "      attack:hasTargetIP ?destIP; " +
    "      attack:pcFrequency ?cnt . " +
    "} " +
    "WHERE { { " +
   "   SELECT ?destIP (MIN(?dateTime) as ?beginDateTime) " +
   "        (MAX (?dateTime) as ?endDateTime) " +
   "        (count(?destIP) as ?cnt) " +
   "        WHERE {?pack rdf:type traffic:PingPacket; " +
   "                   traffic:dateTime ?dateTime; " +
   "                   traffic:hasDestIP ?destIP . " +
   "       } " +
   "       GROUP BY ?destIP  " +
   "       HAVING (count(?destIP) > 0) " +
   " } " +
   "}";

  addCollectionsTime = addCollectionsTime +
                    KButility.execUpdQuery(queryStr, outputFile, false);

    outputFile.println("Adding PingFloodType instances from SmurfPacket...");

    // Retrieve all SmurfPacket instances to same destIPs
    queryStr =
    "PREFIX traffic: " +
    "<" + KButility.URL_PREFIX + "traffic.owl#> " +
    "PREFIX attack: " +
    "<" + KButility.URL_PREFIX + "attack.owl#> " +
    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
    "INSERT " +
     "{" +
    "   _:a rdf:type attack:PacketCollection; " +
    "       attack:beginDate ?beginDateTime; " +
    "       attack:endDate ?endDateTime; " +
     "       attack:pcType traffic:PingFloodType; " +
     "       attack:hasTargetIP ?destIP; " +
     "       attack:pcFrequency ?cnt . " +
     "} " +
     "WHERE { { " +
    "     SELECT ?destIP (MIN(?dateTime) as ?beginDateTime) " +
    "         (MAX (?dateTime) as ?endDateTime) " +
    "         (count(?destIP) as ?cnt) " +
    "         WHERE {?pack rdf:type traffic:SmurfPacket; " +
    "                     traffic:dateTime ?dateTime; " +
    "                     traffic:hasDestIP ?destIP . " +
    "               } " +
    "        GROUP BY ?destIP  " +
    "        HAVING (count(?destIP) > 0) " +
    " } " +
    "}";

  addCollectionsTime = addCollectionsTime +
                    KButility.execUpdQuery(queryStr, outputFile, false);


    }  // end initial try

    catch(Exception e) {
              e.printStackTrace();
    }  // end catch

return addCollectionsTime;
```

```
}   // end function addPingFloods



/******************************************************************/
/*****                                                   *****/
/*****        Add ICMPFlood PacketColletion Instances              *****/
/*****                                                   *****/
/******************************************************************/
public static double addICMPFloods(PrintStream outputFile,
                        double addCollectionsTime) {

try {

   sparqlTime = 0;

   System.out.println("\tAdding ICMP Flood instances...");
   outputFile.println("Adding ICMPFlood instances from MaskPacket...");

   // Retrieve all MaskPacket instances to same destIPs
   String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX attack: " +
   "<" + KButility.URL_PREFIX + "attack.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "INSERT " +
    "{" +
   "   _:a rdf:type attack:PacketCollection; " +
   "      attack:beginDate ?beginDateTime; " +
   "      attack:endDate ?endDateTime; " +
    "      attack:pcType traffic:ICMPFloodType; " +
    "      attack:hasTargetIP ?destIP; " +
    "      attack:pcFrequency ?cnt . " +
    "} " +
    "WHERE { { " +
   "    SELECT ?destIP (MIN(?dateTime) as ?beginDateTime) " +
   "          (MAX (?dateTime) as ?endDateTime) " +
   "          (count(?destIP) as ?cnt) " +
   "          WHERE {?pack rdf:type traffic:MaskPacket; " +
   "                    traffic:dateTime ?dateTime; " +
   "                    traffic:hasDestIP ?destIP . " +
   "          } " +
   "          GROUP BY ?destIP " +
   "       HAVING (count(?destIP) > 0) " +
   " } " +
   "}";

  addCollectionsTime = addCollectionsTime +
                    KButility.execUpdQuery(queryStr, outputFile, false);

   }  // end initial try

   catch(Exception e) {
            e.printStackTrace();
   }  // end catch

return addCollectionsTime;
}   // end function addICMPFloods



/***********************************************************************/
/*****                                              *****/
/*****         Add TCPFlood PacketColletion Instances              *****/
/*****                                              *****/
/***********************************************************************/
public static double addTCPFloods(PrintStream outputFile,
                          double addCollectionsTime) {

try {

   sparqlTime = 0;

   System.out.println("\tAdding TCP Flood instances...");
   outputFile.println("Adding TCPFlood instances from TCPPacket...");

   // Retrieve all TCP Packet instances to same destIP and tcpSynFlag = true
   String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX attack: " +
   "<" + KButility.URL_PREFIX + "attack.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "INSERT " +
    "{" +
   "   _:a rdf:type attack:PacketCollection; " +
   "       attack:beginDate ?beginDateTime; " +
   "       attack:endDate ?endDateTime; " +
    "       attack:pcType traffic:TCPFloodType; " +
    "       attack:hasTargetIP ?destIP; " +
    "       attack:pcFrequency ?cnt . " +
    "} " +
    "WHERE { { " +
   "   SELECT ?destIP " +
   "         (MIN(?dateTime) as ?beginDateTime) " +
   "         (MAX (?dateTime) as ?endDateTime) " +
   "         (count(?destIP) as ?cnt) " +
   "   { " +
   "    SELECT DISTINCT ?packet1 ?destIP ?dateTime " +
   "     { " +
   "       ?packet1 rdf:type traffic:TCPPacket; " +
   "               traffic:dateTime ?dateTime; " +
   "               traffic:hasDestIP ?destIP; " +
   "                      traffic:tcpSynFlag true . " +
   "      { " +
   "        SELECT DISTINCT ?packet2 ?destIP ?dateTime2 " +
   "        { " +
   "          ?packet2 rdf:type traffic:TCPPacket; " +
   "                  traffic:dateTime ?dateTime2; " +
   "                  traffic:hasDestIP ?destIP; " +
   "                         traffic:tcpSynFlag true . " +
   "        } " +
   "      } " +
   "      FILTER ( ?packet1 != ?packet2 ) . " +
   "     } " +
   "   } " +
   "  GROUP BY ?destIP " +
   "  HAVING (count(?destIP) > 0) " +
   " } " +
   "}";

  addCollectionsTime = addCollectionsTime +
                    KButility.execUpdQuery(queryStr, outputFile, false);

   }  // end initial try

   catch(Exception e) {
            e.printStackTrace();
   }  // end catch

return addCollectionsTime;
}   // end function addTCPFloods
```

```
/******************************************************************/
/*****                                                      *****/
/*****      Add AppFlood PacketColletion Instances               *****/
/*****                                                      *****/
/******************************************************************/
public static double addAppFloods(PrintStream outputFile,
                         double addCollectionsTime) {

try {

   sparqlTime = 0;

   System.out.println("\tAdding App Flood instances...");
   outputFile.println("Adding AppFlood instances from AppPacket...");

   // Retrieve all AppPacket instances to same destIP and destPort
   String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX attack: " +
   "<" + KButility.URL_PREFIX + "attack.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "INSERT " +
    "{" +
   "   _:a rdf:type attack:PacketCollection; " +
   "      attack:beginDate ?beginDateTime; " +
   "      attack:endDate ?endDateTime; " +
    "      attack:pcType traffic:AppFloodType; " +
    "      attack:hasTargetIP ?destIP; " +
    "      attack:pcFrequency ?cnt . " +
    "} " +
    "WHERE { { " +
   "   SELECT ?destIP " +
   "        (MIN(?dateTime) as ?beginDateTime) " +
   "        (MAX (?dateTime) as ?endDateTime) " +
   "        (count(?destIP) as ?cnt) " +
   "   { " +
   "      ?packet1 rdf:type traffic:AppPacket; " +
   "              traffic:dateTime ?dateTime; " +
   "              traffic:hasDestIP ?destIP; " +
   "                   traffic:l4DestPort ?l4DestPort . " +
   "     { " +
   "       SELECT ?packet2 ?destIP ?l4DestPort ?dateTime2 " +
   "       { " +
   "         ?packet2 rdf:type traffic:AppPacket; " +
   "                traffic:dateTime ?dateTime2; " +
   "                traffic:hasDestIP ?destIP; " +
   "                     traffic:l4DestPort ?l4DestPort . " +
   "       } " +
   "       GROUP BY ?destIP ?l4DestPort " +
   "     } " +
   "     FILTER ( ?packet1 != ?packet2 ) . " +
   "   } " +
   "  GROUP BY ?destIP ?l4DestPort " +
   "  HAVING (count(?destIP) > 0) " +
   "  ORDER BY ?destIP " +
   " } " +
   "}";

  addCollectionsTime = addCollectionsTime +
                 KButility.execUpdQuery(queryStr, outputFile, false);

   }  // end initial try

   catch(Exception e) {
           e.printStackTrace();
   }  // end catch

return addCollectionsTime;
}   // end function addAppFloods



/*****************************************************************/
/*****                                         *****/
/*****          ADD PingScan PacketColletions Instances          *****/
/*****                                         *****/
/*****************************************************************/
public static double addPingScans(PrintStream outputFile,
                      double addCollectionsTime) {

try {

   sparqlTime = 0;
   String queryStr;
```

```
System.out.println("\tAdding Ping Scan instances...");

// Find all ping scans by comparing appropriate octets for equality
// Class A - first octet 0 - 127
// Class B - first octet 128 - 191
// Class C - first octet 192 - 223

// Class A networks
queryStr =
"PREFIX traffic: " +
"<" + KButility.URL_PREFIX + "traffic.owl#> " +
"PREFIX attack: " +
"<" + KButility.URL_PREFIX + "attack.owl#> " +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
"PREFIX apf: <http://jena.hpl.hp.com/ARQ/property#> " +
"INSERT " +
 "{" +
"   _:a rdf:type attack:PacketCollection; " +
"      attack:beginDate ?beginDateTime; " +
"      attack:endDate ?endDateTime; " +
 "      attack:pcType traffic:PingScanType; " +
 "      attack:hasTargetIP ?nwadd; " +
 "      attack:pcFrequency ?cnt . " +
 "} " +
 "WHERE { { " +
"   SELECT ?nwadd ?IPoctet1 " +
"         (MIN(?dateTime) as ?beginDateTime) " +
"         (MAX (?dateTime) as ?endDateTime) " +
"         (count(?nwadd) as ?cnt) " +
"   { " +
"     SELECT DISTINCT ?packet1 ?ipadd1 ?IPoctet1 ?IPoctet2a " +
"                   ?IPoctet3a ?IPoctet4a ?nwadd ?dateTime" +
"     { " +
"       ?packet1 rdf:type traffic:PingPacket; " +
"              traffic:hasDestIP ?ipadd1; " +
"              traffic:dateTime ?dateTime . " +
"       ?ipadd1  rdf:type traffic:IPaddress; " +
"              traffic:IPoctet1 ?IPoctet1; " +
"              traffic:IPoctet2 ?IPoctet2a; " +
"              traffic:IPoctet3 ?IPoctet3a; " +
"              traffic:IPoctet4 ?IPoctet4a; " +
"              traffic:hasNWIPaddress ?nwadd . " +
"      { " +
"        SELECT DISTINCT ?packet2 ?ipadd2 ?IPoctet1 ?IPoctet2b " +
  "                      ?IPoctet3b ?IPoctet4b ?nwadd2 " +
  "         { " +
  "           ?packet2 rdf:type traffic:PingPacket; " +
  "                  traffic:hasDestIP ?ipadd2; " +
  "                  traffic:dateTime ?dateTime2 . " +
  "           ?ipadd2  rdf:type traffic:IPaddress; " +
  "                  traffic:IPoctet1 ?IPoctet1; " +
  "                  traffic:IPoctet2 ?IPoctet2b; " +
  "                  traffic:IPoctet3 ?IPoctet3b; " +
  "                  traffic:IPoctet4 ?IPoctet4b; " +
  "                  traffic:hasNWIPaddress ?nwadd2 . " +
  "         } " +
  "      } " +
  "     FILTER ( ( ?packet1 != ?packet2 ) &&  " +
  "             ( ?IPoctet1 >= 0 ) &&  " +
  "             ( ?IPoctet1 <= 127 ) ) . " +
  "     } " +
  "   } " +
  "   GROUP BY ?nwadd ?IPoctet1 " +
  " } " +
  "}";

addCollectionsTime = addCollectionsTime +
                  KButility.execUpdQuery(queryStr, outputFile, false);


// Class B networks
  queryStr =
  "PREFIX traffic: " +
  "<" + KButility.URL_PREFIX + "traffic.owl#> " +
  "PREFIX attack: " +
  "<" + KButility.URL_PREFIX + "attack.owl#> " +
  "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
  "PREFIX apf: <http://jena.hpl.hp.com/ARQ/property#> " +
  "INSERT " +
   "{" +
  "   _:a rdf:type attack:PacketCollection; " +
  "       attack:beginDate ?beginDateTime; " +
  "       attack:endDate ?endDateTime; " +
   "       attack:pcType traffic:PingScanType; " +
   "       attack:hasTargetIP ?nwadd; " +
   "       attack:pcFrequency ?cnt . " +
   "} " +
   "WHERE { { " +
  "   SELECT ?nwadd ?IPoctet1 ?IPoctet2 " +
  "        (MIN(?dateTime) as ?beginDateTime) " +
  "        (MAX (?dateTime) as ?endDateTime) " +
  "        (count(?nwadd) as ?cnt) " +
  "   { " +
  "     SELECT DISTINCT ?packet1 ?ipadd1 ?IPoctet1 ?IPoctet2 " +
  "                 ?IPoctet3a ?IPoctet4a ?nwadd ?dateTime" +
  "     { " +
  "       ?packet1 rdf:type traffic:PingPacket; " +
  "              traffic:hasDestIP ?ipadd1; " +
  "              traffic:dateTime ?dateTime . " +
  "       ?ipadd1  rdf:type traffic:IPaddress; " +
  "              traffic:IPoctet1 ?IPoctet1; " +
  "              traffic:IPoctet2 ?IPoctet2; " +
  "              traffic:IPoctet3 ?IPoctet3a; " +
  "              traffic:IPoctet4 ?IPoctet4a; " +
  "              traffic:hasNWIPaddress ?nwadd . " +
  "      { " +
  "        SELECT DISTINCT ?packet2 ?ipadd2 ?IPoctet1 ?IPoctet2 " +
  "                     ?IPoctet3b ?IPoctet4b ?nwadd2 " +
  "        { " +
  "          ?packet2 rdf:type traffic:PingPacket; " +
  "                 traffic:hasDestIP ?ipadd2; " +
  "                 traffic:dateTime ?dateTime2 . " +
  "          ?ipadd2  rdf:type traffic:IPaddress; " +
  "                 traffic:IPoctet1 ?IPoctet1; " +
  "                 traffic:IPoctet2 ?IPoctet2; " +
  "                 traffic:IPoctet3 ?IPoctet3b; " +
  "                 traffic:IPoctet4 ?IPoctet4b; " +
  "                 traffic:hasNWIPaddress ?nwadd2 . " +
  "        } " +
  "      } " +
  "     FILTER ( ( ?packet1 != ?packet2 ) &&  " +
  "            ( ?IPoctet1 >= 128 ) &&  " +
  "            ( ?IPoctet1 <= 191 ) ) . " +
  "     } " +
  "   } " +
  "   GROUP BY ?nwadd ?IPoctet1 ?IPoctet2 " +
  " } " +
  "}";

addCollectionsTime = addCollectionsTime +
            KButility.execUpdQuery(queryStr, outputFile, false);
```

```
// Class C networks
  queryStr =
  "PREFIX traffic: " +
  "<" + KButility.URL_PREFIX + "traffic.owl#> " +
  "PREFIX attack: " +
  "<" + KButility.URL_PREFIX + "attack.owl#> " +
  "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
  "PREFIX apf: <http://jena.hpl.hp.com/ARQ/property#> " +
  "INSERT " +
   "{" +
  "   _:a rdf:type attack:PacketCollection; " +
  "      attack:beginDate ?beginDateTime; " +
  "      attack:endDate ?endDateTime; " +
   "      attack:pcType traffic:PingScanType; " +
  "      attack:hasTargetIP ?nwadd; " +
   "      attack:pcFrequency ?cnt . " +
   "} " +
   "WHERE { { " +
  "   SELECT ?nwadd ?IPoctet1 ?IPoctet2 ?IPoctet3 " +
  "         (MIN(?dateTime) as ?beginDateTime) " +
  "         (MAX (?dateTime) as ?endDateTime) " +
  "         (count(?nwadd) as ?cnt) " +
  "   { " +
  "    SELECT DISTINCT ?packet1 ?ipadd1 ?IPoctet1 ?IPoctet2 " +
  "                 ?IPoctet3 ?IPoctet4a ?nwadd ?dateTime" +
  "    { " +
  "      ?packet1 rdf:type traffic:PingPacket; " +
  "             traffic:hasDestIP ?ipadd1; " +
  "             traffic:dateTime ?dateTime . " +
  "      ?ipadd1  rdf:type traffic:IPaddress; " +
  "             traffic:IPoctet1 ?IPoctet1; " +
  "             traffic:IPoctet2 ?IPoctet2; " +
  "             traffic:IPoctet3 ?IPoctet3; " +
  "             traffic:IPoctet4 ?IPoctet4a; " +
  "             traffic:hasNWIPaddress ?nwadd . " +
  "     { " +
  "       SELECT DISTINCT ?packet2 ?ipadd2 ?IPoctet1 ?IPoctet2 " +
  "                     ?IPoctet3 ?IPoctet4b ?nwadd2 " +
  "       { " +
  "         ?packet2 rdf:type traffic:PingPacket; " +
  "                traffic:hasDestIP ?ipadd2; " +
  "                traffic:dateTime ?dateTime2 . " +
  "         ?ipadd2  rdf:type traffic:IPaddress; " +
  "                traffic:IPoctet1 ?IPoctet1; " +
  "                  traffic:IPoctet2 ?IPoctet2; " +
  "                  traffic:IPoctet3 ?IPoctet3; " +
  "                  traffic:IPoctet4 ?IPoctet4b; " +
  "                  traffic:hasNWIPaddress ?nwadd2 . " +
  "         } " +
  "      } " +
  "     FILTER ( ( ?packet1 != ?packet2 ) &&  " +
  "             ( ?IPoctet1 >= 192 ) &&  " +
  "             ( ?IPoctet1 <= 223 ) ) . " +
  "     } " +
  "   } " +
  "   GROUP BY ?nwadd ?IPoctet1 ?IPoctet2 ?IPoctet3 " +
  " } " +
  "}";

addCollectionsTime = addCollectionsTime +
                   KButility.execUpdQuery(queryStr, outputFile, false);


// Add Host IP addresses into NWaddressScanned class for each
//      host whose network was scanned with a PingScan attack.
System.out.println("\tAdding Host IP address into NWaddressScanned class...");
  outputFile.println("Adding Host IP address into NWaddressScanned class...");

  queryStr =
  "PREFIX traffic: " +
  "<" + KButility.URL_PREFIX + "traffic.owl#> " +
  "PREFIX attack: " +
  "<" + KButility.URL_PREFIX + "attack.owl#> " +
  "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
  "INSERT " +
   "{" +
  "   ?hostadd rdf:type traffic:NWaddressScanned; " +
  "           traffic:IPoctet1 ?IPoctet1; " +
  "           traffic:IPoctet2 ?IPoctet2; " +
   "           traffic:IPoctet3 ?IPoctet3; " +
   "           traffic:IPoctet4 ?IPoctet4 . " +
   "} " +
   "WHERE { { " +
  "   SELECT ?ipadd ?IPoctet1 ?IPoctet2 " +
  "                 ?IPoctet3 ?IPoctet4 " +
  "   { " +
  "      ?ipadd  rdf:type traffic:IPaddress; " +
  "              traffic:IPoctet1 ?IPoctet1; " +
  "              traffic:IPoctet2 ?IPoctet2; " +
  "              traffic:IPoctet3 ?IPoctet3; " +
  "              traffic:IPoctet4 ?IPoctet4; " +
  "              traffic:hasNWIPaddress ?ipadd1 . " +
  "    { " +
  "     SELECT ?packet1 ?ipadd1 " +
  "     { " +
  "       ?packet1 rdf:type attack:PingScan; " +
  "               attack:hasTargetIP ?ipadd1 . " +
  "     } " +
  "    } " +
  "   FILTER  ( ( ?IPoctet4 != 0 ) ) " +
  "   } " +
  " } " +
  "LET (?hostadd := ?ipadd) . " +
  "}";

 addCollectionsTime = addCollectionsTime +
                   KButility.execUpdQuery(queryStr, outputFile, false);

  }  // end initial try

  catch(Exception e) {
            e.printStackTrace();
  }  // end catch

return addCollectionsTime;
}   // end function addPingScans



/****************************************************************/
/*****                                               *****/
/*****         ADD PortScan PacketColletions Instances              *****/
/*****                                               *****/
/****************************************************************/
public static double addPortScans(PrintStream outputFile,
                          double addCollectionsTime) {

try {

            sparqlTime = 0;

  System.out.println("\tAdding Port Scan instances...");
  outputFile.println("Adding PortScan instances of multiple ports to same node...");
```

```
// Retrieve all L4Packet instances to same destIPs with different Ports
String queryStr =
"PREFIX traffic: " +
"<" + KButility.URL_PREFIX + "traffic.owl#> " +
"PREFIX attack: " +
"<" + KButility.URL_PREFIX + "attack.owl#> " +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
"INSERT " +
 "{" +
"   _:a rdf:type attack:PacketCollection; " +
"      attack:beginDate ?beginDateTime; " +
"      attack:endDate ?endDateTime; " +
 "      attack:pcType traffic:PortScanType; " +
 "      attack:hasTargetIP ?destIP; " +
 "      attack:pcFrequency ?cnt . " +
 "} " +
 "WHERE { { " +
"   SELECT DISTINCT ?packet1 ?destIP " +
"         (MIN(?dateTime) as ?beginDateTime) " +
"         (MAX (?dateTime) as ?endDateTime) " +
"         (count(?destIP) as ?cnt) " +
"   { " +
"      ?packet1 rdf:type traffic:L4Packet; " +
"             traffic:dateTime ?dateTime; " +
"             traffic:hasDestIP ?destIP; " +
"                   traffic:l4DestPort ?l4DestPort1 . " +
"     { " +
"       SELECT ?packet2 ?destIP ?l4DestPort2 ?dateTime2 " +
"       { " +
"         ?packet2 rdf:type traffic:L4Packet; " +
"                 traffic:dateTime ?dateTime2; " +
"                 traffic:hasDestIP ?destIP; " +
"                       traffic:l4DestPort ?l4DestPort2 . " +
"       } " +
"       GROUP BY ?destIP " +
"     } " +
"     FILTER ( ( ?packet1 != ?packet2) &&  " +
"            ( ?l4DestPort1 != ?l4DestPort2 ) ) . " +
"   } " +
"  GROUP BY ?destIP " +
"  HAVING (count(?destIP) > 0) " +
" } " +
"}";
```

```
addCollectionsTime = addCollectionsTime +
                KButility.execUpdQuery(queryStr, outputFile, false);

outputFile.println("Adding PortScan instances from SynPacket...");

 // Retrieve all SynPacket instances to same destIPs
 queryStr =
 "PREFIX traffic: " +
 "<" + KButility.URL_PREFIX + "traffic.owl#> " +
 "PREFIX attack: " +
 "<" + KButility.URL_PREFIX + "attack.owl#> " +
 "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
 "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
 "INSERT " +
  "{" +
 "   _:a rdf:type attack:PacketCollection; " +
 "      attack:beginDate ?beginDateTime; " +
 "      attack:endDate ?endDateTime; " +
  "      attack:pcType traffic:PortScanType; " +
  "      attack:hasTargetIP ?destIP; " +
  "      attack:pcFrequency ?cnt . " +
  "} " +
  "WHERE { { " +
 "    SELECT ?destIP (MIN(?dateTime) as ?beginDateTime) " +
 "          (MAX (?dateTime) as ?endDateTime) " +
 "          (count(?destIP) as ?cnt) " +
 "          WHERE {?pack rdf:type traffic:SynPacket; " +
 "                    traffic:dateTime ?dateTime; " +
 "                    traffic:hasDestIP ?destIP . " +
 "          } " +
 "      GROUP BY ?destIP " +
 "      HAVING (count(?destIP) > 0) " +
 " } " +
 "}";

addCollectionsTime = addCollectionsTime +
                KButility.execUpdQuery(queryStr, outputFile, false);


 outputFile.println("Adding PortScan instances from FinPacket...");
 // Retrieve all FinPacket instances to same destIPs
 // FinPacket instances are TCP packets that only have FIN flag set
 queryStr =
 "PREFIX traffic: " +
  "<" + KButility.URL_PREFIX + "traffic.owl#> " +
  "PREFIX attack: " +
  "<" + KButility.URL_PREFIX + "attack.owl#> " +
  "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
  "INSERT " +
   "{" +
  "   _:a rdf:type attack:PacketCollection; " +
  "       attack:beginDate ?beginDateTime; " +
  "       attack:endDate ?endDateTime; " +
   "       attack:pcType traffic:PortScanType; " +
   "       attack:hasTargetIP ?destIP; " +
   "       attack:pcFrequency ?cnt . " +
   "} " +
   "WHERE { { " +
  "     SELECT ?destIP (MIN(?dateTime) as ?beginDateTime) " +
  "            (MAX (?dateTime) as ?endDateTime) " +
  "            (count(?destIP) as ?cnt) " +
  "            WHERE {?pack rdf:type traffic:FinPacket; " +
  "                       traffic:dateTime ?dateTime; " +
  "                       traffic:hasDestIP ?destIP . " +
  "            } " +
  "        GROUP BY ?destIP " +
  "        HAVING (count(?destIP) > 0) " +
  " } " +
  "}";

addCollectionsTime = addCollectionsTime +
                  KButility.execUpdQuery(queryStr, outputFile, false);


  outputFile.println("Adding PortScan instances from NullPacket...");
  // Retrieve all NullPacket instances to same destIPs
  queryStr =
  "PREFIX traffic: " +
  "<" + KButility.URL_PREFIX + "traffic.owl#> " +
  "PREFIX attack: " +
  "<" + KButility.URL_PREFIX + "attack.owl#> " +
  "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
  "INSERT " +
   "{" +
  "   _:a rdf:type attack:PacketCollection; " +
  "       attack:beginDate ?beginDateTime; " +
  "       attack:endDate ?endDateTime; " +
    "       attack:pcType traffic:PortScanType; " +
    "       attack:hasTargetIP ?destIP; " +
    "       attack:pcFrequency ?cnt . " +
    "} " +
    "WHERE { { " +
   "     SELECT ?destIP (MIN(?dateTime) as ?beginDateTime) " +
   "           (MAX (?dateTime) as ?endDateTime) " +
   "           (count(?destIP) as ?cnt) " +
   "           WHERE {?pack rdf:type traffic:NullPacket; " +
   "                     traffic:dateTime ?dateTime; " +
   "                     traffic:hasDestIP ?destIP . " +
   "           } " +
   "       GROUP BY ?destIP " +
   "       HAVING (count(?destIP) > 0) " +
   " } " +
   "}";

  addCollectionsTime = addCollectionsTime +
                 KButility.execUpdQuery(queryStr, outputFile, false);

   }  // end initial try

   catch(Exception e) {
            e.printStackTrace();
   }  // end catch

return addCollectionsTime;
}   // end function addPortScans


/******************************************************************/
/*****                                                *****/
/*****          Add PingScan PacketColletion Instances              *****/
/*****        Add for nodes where network instance exists           *****/
/*****                                                *****/
/******************************************************************/
public static double addNodePingScans(PrintStream outputFile,
                          double addCollectionsTime) {

try {

            sparqlTime = 0;

   // Class A networks
```

```
        System.out.println("\tAdding Ping Scans for nodes from network class A scan
instances...");
        outputFile.println("Adding PingScan instances for nodes from network class A
scans...");

        String queryStr =
        "PREFIX traffic: " +
        "<" + KButility.URL_PREFIX + "traffic.owl#> " +
        "PREFIX attack: " +
        "<" + KButility.URL_PREFIX + "attack.owl#> " +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX apf: <http://jena.hpl.hp.com/ARQ/property#> " +
        "INSERT " +
         "{" +
        "   _:a rdf:type attack:PacketCollection; " +
        "      attack:beginDate ?beginDateTime; " +
        "      attack:endDate ?endDateTime; " +
         "      attack:pcType attack:PingScanType; " +
         "      attack:hasTargetIP ?nodeadd; " +
         "      attack:pcFrequency ?pcFreq . " +
         "} " +
         "WHERE { { " +
        "   SELECT ?beginDateTime ?endDateTime ?pcFreq " +
        "         ?nodeadd " +
        "   { " +
        "    SELECT DISTINCT ?packet1 ?ipadd1 ?beginDateTime ?endDateTime " +
        "                 ?pcFreq ?IPoctet1 " +
        "    { " +
        "      ?packet1 rdf:type attack:PingScan; " +
        "             attack:hasTargetIP ?ipadd1; " +
        "             attack:beginDate ?beginDateTime; " +
        "             attack:endDate ?endDateTime; " +
        "             attack:pcFrequency ?pcFreq . " +
        "      ?ipadd1  rdf:type traffic:IPaddress; " +
        "             traffic:IPoctet1 ?IPoctet1; " +
        "             traffic:IPoctet2 0; " +
        "             traffic:IPoctet3 0; " +
        "             traffic:IPoctet4 0 . " +
        "     { " +
        "       SELECT DISTINCT ?nodeadd ?IPoctet1 " +
        "        { " +
        "         ?nodeadd  rdf:type traffic:IPaddress; " +
        "                traffic:IPoctet1 ?IPoctet1; " +
        "                traffic:IPoctet2 ?IPoctet2b; " +
        "                  traffic:IPoctet3 ?IPoctet3b; " +
        "                  traffic:IPoctet4 ?IPoctet4b . " +
        "         } " +
        "      } " +
        "    FILTER ( ( ?ipadd1 != ?nodeadd ) ) . " +
        "    } " +
        "   } " +
        " } " +
        "}";

      addCollectionsTime = addCollectionsTime +
                      KButility.execUpdQuery(queryStr, outputFile, false);


      // Class B networks
        System.out.println("\tAdding Ping Scans for nodes from network class B scan
instances...");
        outputFile.println("Adding PingScan instances for nodes from network class B
scans...");

        queryStr =
        "PREFIX traffic: " +
        "<" + KButility.URL_PREFIX + "traffic.owl#> " +
        "PREFIX attack: " +
        "<" + KButility.URL_PREFIX + "attack.owl#> " +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX apf: <http://jena.hpl.hp.com/ARQ/property#> " +
        "INSERT " +
         "{" +
        "   _:a rdf:type attack:PacketCollection; " +
        "      attack:beginDate ?beginDateTime; " +
        "      attack:endDate ?endDateTime; " +
         "      attack:pcType attack:PingScanType; " +
         "      attack:hasTargetIP ?nodeadd; " +
         "      attack:pcFrequency ?pcFreq . " +
         "} " +
         "WHERE { { " +
        "    SELECT DISTINCT ?packet1 ?ipadd1 ?beginDateTime ?endDateTime " +
        "                  ?pcFreq ?IPoctet1 ?IPoctet2 " +
        "   { " +
        "    SELECT DISTINCT ?packet1 ?ipadd1 ?IPoctet1 ?IPoctet2 " +
        "                  ?IPoctet3 ?IPoctet4a ?nwadd ?dateTime" +
        "    { " +
        "      ?packet1 rdf:type attack:PingScan; " +
        "            attack:hasTargetIP ?ipadd1; " +
        "            attack:beginDate ?beginDateTime; " +
        "            attack:endDate ?endDateTime; " +
        "            attack:pcFrequency ?pcFreq . " +
        "      ?ipadd1  rdf:type traffic:IPaddress; " +
        "            traffic:IPoctet1 ?IPoctet1; " +
        "            traffic:IPoctet2 ?IPoctet2; " +
        "            traffic:IPoctet3 0; " +
        "            traffic:IPoctet4 0 . " +
        "     { " +
        "       SELECT DISTINCT ?nodeadd ?IPoctet1 ?IPoctet2 " +
        "        { " +
        "         ?nodeadd  rdf:type traffic:IPaddress; " +
        "               traffic:IPoctet1 ?IPoctet1; " +
        "               traffic:IPoctet2 ?IPoctet2; " +
        "               traffic:IPoctet3 ?IPoctet3b; " +
        "               traffic:IPoctet4 ?IPoctet4b . " +
        "        } " +
        "     } " +
        "    FILTER ( ( ?ipadd1 != ?nodeadd ) ) . " +
        "    } " +
        "  } " +
        " } " +
        "}";

      addCollectionsTime = addCollectionsTime +
                      KButility.execUpdQuery(queryStr, outputFile, false);


      // Class C networks
        System.out.println("\tAdding Ping Scans for nodes from network class C scan
instances...");
        outputFile.println("Adding PingScan instances for nodes from network class C
scans...");

        queryStr =
        "PREFIX traffic: " +
        "<" + KButility.URL_PREFIX + "traffic.owl#> " +
        "PREFIX attack: " +
        "<" + KButility.URL_PREFIX + "attack.owl#> " +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX apf: <http://jena.hpl.hp.com/ARQ/property#> " +
        "INSERT " +
         "{" +
        "   _:a rdf:type attack:PacketCollection; " +
        "      attack:beginDate ?beginDateTime; " +
        "      attack:endDate ?endDateTime; " +
         "      attack:pcType attack:PingScanType; " +
         "      attack:hasTargetIP ?nodeadd; " +
         "      attack:pcFrequency ?pcFreq . " +
         "} " +
         "WHERE { { " +
        "    SELECT DISTINCT ?packet1 ?ipadd1 ?beginDateTime ?endDateTime " +
        "                ?pcFreq ?IPoctet1 ?IPoctet2 ?IPoctet3 " +
        "   { " +
        "    SELECT DISTINCT ?packet1 ?ipadd1 ?IPoctet1 ?IPoctet2 " +
        "                ?IPoctet3 ?IPoctet4a ?nwadd ?dateTime" +
        "    { " +
        "      ?packet1 rdf:type attack:PingScan; " +
        "            attack:hasTargetIP ?ipadd1; " +
        "            attack:beginDate ?beginDateTime; " +
        "            attack:endDate ?endDateTime; " +
        "            attack:pcFrequency ?pcFreq . " +
        "      ?ipadd1  rdf:type traffic:IPaddress; " +
        "            traffic:IPoctet1 ?IPoctet1; " +
        "            traffic:IPoctet2 ?IPoctet2; " +
        "            traffic:IPoctet3 ?IPoctet3; " +
        "            traffic:IPoctet4 0 . " +
        "     { " +
        "       SELECT DISTINCT ?nodeadd ?IPoctet1 ?IPoctet2 ?IPoctet3 " +
        "        { " +
        "          ?nodeadd  rdf:type traffic:IPaddress; " +
        "                traffic:IPoctet1 ?IPoctet1; " +
        "                traffic:IPoctet2 ?IPoctet2; " +
        "                traffic:IPoctet3 ?IPoctet3; " +
        "                traffic:IPoctet4 ?IPoctet4b . " +
        "        } " +
        "     } " +
        "    FILTER ( ( ?ipadd1 != ?nodeadd ) ) . " +
        "    } " +
        "   } " +
        " } " +
        "}";

      addCollectionsTime = addCollectionsTime +
                      KButility.execUpdQuery(queryStr, outputFile, false);

        }  // end initial try
```

```
      catch(Exception e) {
                e.printStackTrace();
      }  // end catch

   return addCollectionsTime;
   }   // end function addNodePingScans


 }   // end class PacketCollections
```

C.2 Java File to Create Traffic Stream Instances

```
/*******************************************************************/
/*                                                    */
/* Author: Lisa Frye                                          */
/* Date: January 2011                                          */
/* Filename: TrafficStreams.java                                    */
/*                                                    */
/* Description: This file contains functions to execute SPARQL      */
/*             queries against the KB and add instances for        */
/*             traffic streams based on results from the queries   */
/*             to the KB.                                  */
/* API: this program uses the Jena ontology API.                  */
/*                                                    */
/*******************************************************************/


// imports for Jena API
import com.hp.hpl.jena.update.GraphStore;
import com.hp.hpl.jena.update.GraphStoreFactory;
import com.hp.hpl.jena.update.UpdateAction;
import com.hp.hpl.jena.update.UpdateFactory;
import com.hp.hpl.jena.update.UpdateProcessor;
import com.hp.hpl.jena.update.UpdateRequest;
import com.hp.hpl.jena.update.UpdateExecutionFactory;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.Syntax;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;

// general imports
import java.lang.*;
import java.io.*;
import java.util.*;
import java.text.DecimalFormat;
```

```
public class TrafficStreams {


 /*******************************************************************/
 /*****                                              *****/
 /*****              ADD TCP Stream Instances                  *****/
 /*****                                              *****/
 /*******************************************************************/
 public static double addTCPStreams(PrintStream outputFile, double addStreamsTime) {

try {

  // build a query string to insert all triples selected that are
 // TCP packets with unique src and dest IP and src and dest port numbers.
 String queryStr =
    "PREFIX traffic: " +
    "<" + KButility.URL_PREFIX + "traffic.owl#> " +
    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
    "PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#> " +
    "INSERT " +
    "{ " +
    "   ?stream rdf:type traffic:TCPStream; " +
    "          traffic:protocol \"TCP\"; " +
    "          traffic:startTime ?dateTime; " +
    "          traffic:endTime ?dateTime; " +
    "          traffic:hasNode1MAC ?srcMAC; " +
    "          traffic:hasNode2MAC ?destMAC; " +
    "          traffic:hasNode1IP ?srcIP; " +
    "          traffic:hasNode2IP ?destIP; " +
    "          traffic:node1Port ?l4SrcPort; " +
    "          traffic:node2Port ?l4DestPort . " +
    "   ?srcIP  traffic:hasTCPStreamWith ?destIP; " +
    "} " +
    "WHERE  { { " +
    "   SELECT DISTINCT ?packet ?dateTime ?srcMAC ?destMAC " +
    "            ?srcIP ?destIP ?l4SrcPort ?l4DestPort { " +
    "      ?packet rdf:type traffic:TCPPacket; " +
    "             traffic:dateTime ?dateTime; " +
    "                        traffic:hasSrcMAC ?srcMAC; " +
    "                        traffic:hasDestMAC ?destMAC; " +
    "                        traffic:hasSrcIP ?srcIP; " +
    "                        traffic:hasDestIP ?destIP; " +
    "                        traffic:l4SrcPort ?l4SrcPort; " +
    "                     traffic:l4DestPort ?l4DestPort . " +
    "    } " +
    " } " +
    "LET (?stream := ?packet) . " +
    "}";

   outputFile.println("Adding TCP Streams...");
   addStreamsTime = addStreamsTime + KButility.execUpdQuery(queryStr, outputFile,
false);

         }  // end initial try

         catch(Exception e) {
                 e.printStackTrace();
         }  // end catch

 return addStreamsTime;
 }   // end function addTCPStreams


 /****************************************************************/
 /*****                                            *****/
 /*****              ADD UDP Stream Instances                 *****/
 /*****                                            *****/
 /****************************************************************/
 public static double addUDPStreams(PrintStream outputFile, double addStreamsTime)
{

 try {

  // build a query string to insert all triples selected that are
  // UDP packets with unique src and dest IP and src and dest port numbers.
  String queryStr =
    "PREFIX traffic: " +
    "<" + KButility.URL_PREFIX + "traffic.owl#> " +
    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
    "PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#> " +
    "INSERT " +
    "{ " +
    "   ?stream rdf:type traffic:UDPStream; " +
    "          traffic:protocol \"UDP\"; " +
    "          traffic:startTime ?dateTime; " +
    "          traffic:endTime ?dateTime; " +
    "          traffic:hasNode1MAC ?srcMAC; " +
    "          traffic:hasNode2MAC ?destMAC; " +
   "          traffic:hasNode1IP ?srcIP; " +
   "          traffic:hasNode2IP ?destIP; " +
   "          traffic:node1Port ?l4SrcPort; " +
   "          traffic:node2Port ?l4DestPort . " +
   "} " +
   "WHERE  { { " +
   "   SELECT DISTINCT ?packet ?dateTime ?srcMAC ?destMAC " +
   "            ?srcIP ?destIP ?l4SrcPort ?l4DestPort { " +
   "      ?packet rdf:type traffic:UDPPacket; " +
   "            traffic:dateTime ?dateTime; " +
   "                       traffic:hasSrcMAC ?srcMAC; " +
   "                       traffic:hasDestMAC ?destMAC; " +
   "                       traffic:hasSrcIP ?srcIP; " +
   "                       traffic:hasDestIP ?destIP; " +
   "                       traffic:l4SrcPort ?l4SrcPort; " +
   "                    traffic:l4DestPort ?l4DestPort . " +
   "   } " +
   " } " +
   "LET (?stream := ?packet) . " +
   "}";

  outputFile.println("Adding UDP Streams...");
  addStreamsTime = addStreamsTime + KButility.execUpdQuery(queryStr, outputFile,
false);

 }  // end initial try

 catch(Exception e) {
         e.printStackTrace();
 }  // end catch

 return addStreamsTime;
 }   // end function addUDPStreams



 /****************************************************************/
 /*****                                                   *****/
 /*****              ADD ICMP Stream Instances                  *****/
 /*****                                                   *****/
 /****************************************************************/
 public static double addICMPStreams(PrintStream outputFile, double addStreamsTime)
{

 try {
 // build a query string to insert all triples selected that are
 // ICMP packets with unique src and dest IP and src and dest port numbers.
 String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#> " +
   "INSERT " +
   "{ " +
   "   ?stream rdf:type traffic:ICMPStream; " +
   "         traffic:protocol \"ICMP\"; " +
   "         traffic:startTime ?dateTime; " +
   "         traffic:endTime ?dateTime; " +
   "         traffic:hasNode1MAC ?srcMAC; " +
   "         traffic:hasNode2MAC ?destMAC; " +
   "         traffic:hasNode1IP ?srcIP; " +
   "         traffic:hasNode2IP ?destIP . " +
   "} " +
   "WHERE  { { " +
   "   SELECT DISTINCT ?packet ?dateTime ?srcMAC ?destMAC ?srcIP ?destIP  { "+
   "      ?packet rdf:type traffic:ICMPPacket; " +
   "            traffic:dateTime ?dateTime; " +
   "                      traffic:hasSrcMAC ?srcMAC; " +
   "                      traffic:hasDestMAC ?destMAC; " +
   "                      traffic:hasSrcIP ?srcIP; " +
   "                      traffic:hasDestIP ?destIP . " +
   "    } " +
   " } " +
   "LET (?stream := ?packet) . " +
   "}";

  outputFile.println("Adding ICMP Streams...");
  addStreamsTime = addStreamsTime + KButility.execUpdQuery(queryStr, outputFile,
false);

         }  // end initial try

         catch(Exception e) {
                  e.printStackTrace();
         }  // end catch


 return addStreamsTime;
```

```
}   // end function addICMPStreams



/*****************************************************************/
/*****                                          *****/
/*****              ADD L3 Stream Instances               *****/
/*****                                          *****/
/*****************************************************************/
public static double addL3Streams(PrintStream outputFile, double addStreamsTime) {

try {

 // build a query string to insert all triples selected that are
 // L3 packets with unique src and dest IP.
 String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#> " +
   "INSERT " +
   "{ " +
   "   ?stream rdf:type traffic:L3Stream; " +
   "          traffic:protocol \"IP\"; " +
   "          traffic:startTime ?dateTime; " +
   "          traffic:endTime ?dateTime; " +
   "          traffic:hasNode1MAC ?srcMAC; " +
   "          traffic:hasNode2MAC ?destMAC; " +
   "          traffic:hasNode1IP ?srcIP; " +
   "          traffic:hasNode2IP ?destIP . " +
   "} " +
   "WHERE  { { " +
   "   SELECT DISTINCT ?packet ?dateTime ?srcMAC ?destMAC ?srcIP ?destIP  { "+
   "      ?packet rdf:type traffic:IPPacket; " +
   "             traffic:dateTime ?dateTime; " +
   "                     traffic:hasSrcMAC ?srcMAC; " +
   "                     traffic:hasDestMAC ?destMAC; " +
   "                     traffic:hasSrcIP ?srcIP; " +
   "                     traffic:hasDestIP ?destIP . " +
   "    } " +
   " } " +
   "LET (?stream := ?packet) . " +
   "}";
```

```
    outputFile.println("Adding L3 Streams...");
    addStreamsTime = addStreamsTime + KButility.execUpdQuery(queryStr, outputFile,
false);

   }  // end initial try

   catch(Exception e) {
          e.printStackTrace();
   }  // end catch


  return addStreamsTime;
  }   // end function addL3Streams


  /********************************************************************/
  /*****                                                  *****/
  /*****                ADD ARP Stream Instances                    *****/
  /*****                                                  *****/
  /********************************************************************/
  public static double addARPStreams(PrintStream outputFile, double addStreamsTime) {

  try {

   // build a query string to insert all triples selected that are
   // ICMP packets with unique src and dest IP and src and dest port
   // numbers.
   String queryStr =
     "PREFIX traffic: " +
     "<" + KButility.URL_PREFIX + "traffic.owl#> " +
     "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
     "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
     "PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#> " +
     "INSERT " +
     "{ " +
     "   ?stream rdf:type traffic:L2Stream; " +
     "          traffic:protocol \"ARP\"; " +
     "          traffic:startTime ?dateTime; " +
     "          traffic:endTime ?dateTime; " +
     "          traffic:hasNode1MAC ?srcMAC; " +
     "          traffic:hasNode2MAC ?destMAC . " +
     "} " +
     "WHERE  { { " +
     "   SELECT DISTINCT ?packet ?dateTime ?srcMAC ?destMAC { " +
     "      ?packet rdf:type traffic:L2Packet; " +
     "              traffic:dateTime ?dateTime; " +
     "                             traffic:srcMAC ?srcMAC; " +
     "                          traffic:destMAC ?destMAC . " +
     "     FILTER NOT EXISTS { ?packet rdf:type traffic:TCPPacket . } " +
     "     FILTER NOT EXISTS { ?packet rdf:type traffic:UDPPacket . } " +
     "     FILTER NOT EXISTS { ?packet rdf:type traffic:ICMPPacket . } " +
     "     FILTER NOT EXISTS { ?packet rdf:type traffic:IPPacket . } " +
     "    } " +
     " } " +
     "LET (?stream := ?packet) . " +
     "}";

    outputFile.println("Adding ICMP Streams...");
    addStreamsTime = addStreamsTime + KButility.execUpdQuery(queryStr, outputFile,
false);

   }  // end initial try

   catch(Exception e) {
          e.printStackTrace();
   }  // end catch


  return addStreamsTime;
  }   // end function addARPStreams


}   // end class TrafficStreams
```

C.3 Java File to Create Attack Instances from Alerts

```
/*****************************************************************/
/*                                                   */
/* Author: Lisa Frye                                          */
/* Date: March 2011                                            */
/* Filename: AlertAttacks.java                                    */
/*                                                   */
/* Description: This file contains functions to execute SPARQL      */
/*             queries against the KB and add instances for        */
/*             simple attacks based on results from the queries    */
/*             to the alert classes of the traffic ontology in     */
/*             in the KB.                                  */
/* API: this program uses the Jena ontology API.                  */
/*                                                   */
/*****************************************************************/


// imports for Jena API
import com.hp.hpl.jena.update.GraphStore;
import com.hp.hpl.jena.update.GraphStoreFactory;
import com.hp.hpl.jena.update.UpdateAction;
import com.hp.hpl.jena.update.UpdateFactory;
import com.hp.hpl.jena.update.UpdateProcessor;
import com.hp.hpl.jena.update.UpdateRequest;
import com.hp.hpl.jena.update.UpdateExecutionFactory;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.Syntax;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;

// general imports
import java.lang.*;
import java.io.*;
import java.util.*;
```

```
public class AlertAttacks {


 /********************************************************************/
 /*****                                                        *****/
 /*****              Add All Alert Attack Instances               *****/
 /*****                                                        *****/
 /********************************************************************/
 public static double addAllAlertAttacks(PrintStream outputFile,
                                  double addAlertAttsTime) {

 try {

   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "DoS",
                        "aClassification", "Denial of Service",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "UnauthAccess",
                        "aClassification",
                        "Attempt to Login By a Default Username and Password",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "UnauthAccess",
                        "aClassification", "root login attempt",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "UnauthAccess",
                        "aClassification",
                        "Attempted Login Using a Suspicious Username was Detected",
                         addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "UserPG",
                        "aClassification", "User Privilege Gain",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "AdminPG",
                        "aClassification", "Administrator Privilege Gain",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "InfoLeak",
                        "Information Leak", "aClassification",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "InfoLeak",
                        "aClassification",
                        "Sensitive Data was Transmitted Across the Network",
                        addAlertAttsTime);
   addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "InfoLeak",
                        "aClassification",
                        "Inappropriate Content was Detected",
                        addAlertAttsTime);
```

```
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "NodeInfo",
                  "aDescription", "ICMP Address Mask Request",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "NodeInfo",
                  "aClassification",
                  "A Client was Using an Unusual Port",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "NodeInfo",
                  "aClassification",
                  "Detection of a Non-Standard Porotcol or Event",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "TCPInfo",
                  "aClassification",
                  "TCP Connection was Detected",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "RPC",
                  "aClassification",  "Decode of an RPC Query",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "ExecCode",
                  "aClassification",
                  "Executable Code was Detected",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "ExecCode",
                  "aClassification",
                  "A Suspicious String was Detected",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "WebServer",
                  "aClassification",
                  "Access to a Potentially Vulnerable Web Application",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "WebServer",
                  "aClassification", "Web Application Attack",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "SystemCall",
                  "aClassification", "A System Call was Detected",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "SendFile",
                  "aClassification",
                  "A Suspicious Filename was Detected",
                  addAlertAttsTime);
addAlertAttsTime = AlertAttacks.addAlertAttacks(outputFile, "SendFile",
                  "aClassification",
                  "A Network Trojan was Detected",
                  addAlertAttsTime);
```

```
  }  // end initial try

 catch(Exception e) {
         e.printStackTrace();
 }  // end catch

return addAlertAttsTime;
}   // end function addAllAlertAttacks



/********************************************************************/
/*****                                               *****/
/*****              Add Alert Attack Instances                *****/
/*****                                               *****/
/********************************************************************/
/* className - the name of the class in the ontology to add the instances */
/* field - the name of the field in the class to perform the regExp match */
/* regExp - the string to search for in the regular expression in query   */
/********************************************************************/
public static double addAlertAttacks(PrintStream outputFile, String className,
                             String field, String regExp,
                             double addAlertAttsTime) {

try {

   outputFile.println("Adding Attacks from Alerts to " + className +
                  " for regexp - " + regExp + "!");

 // Build the query string
 String queryStr =
    "PREFIX attack: " +
    "<" + KButility.URL_PREFIX + "attack.owl#> " +
    "PREFIX traffic: " +
    "<" + KButility.URL_PREFIX + "traffic.owl#> " +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
    "INSERT " +
    "{ " +
    "   ?attack rdf:type attack:" + className + "; " +
    "         attack:attBeginDate ?aDateTime; " +
    "         attack:attEndDate ?aDateTime; " +
    "         attack:description ?aDesc; " +
    "         attack:targetAddress ?aDestIP . " +
    "} " +
    "WHERE  { { " +
      "   SELECT ?alert ?aDateTime ?aDesc ?aDestIP " +
      "   { " +
      "       ?alert rdf:type traffic:Alert; " +
      "               traffic:aDateTime ?aDateTime; " +
      "               traffic:aDescription ?aDesc; " +
      "                           traffic:aClassification ?aClassification . " +
      "     OPTIONAL { ?alert traffic:aDestIP ?aDestIP . } . " +
      "     FILTER REGEX(\"" + field + "\"," +" \"" + regExp + "\", \"i\") . " +
      "   } " +
      " } " +
      "LET (?attack := ?alert) . " +
      "}";

    addAlertAttsTime = addAlertAttsTime + KButility.execUpdQuery(queryStr,
outputFile, false);

  }  // end initial try

   catch(Exception e) {
           e.printStackTrace();
   }  // end catch

  return addAlertAttsTime;
  }   // end function addAlertAttacks


}   // end class AlertAttacks
```

C.4 Java File to Create Some Simple Attack Instances

```
/*****************************************************************/
/*                                              */
/* Author: Lisa Frye                                    */
/* Date: March 2011                                      */
/* Filename: SimpleAttacks.java                               */
/*                                              */
/* Description: This file contains functions to execute SPARQL      */
/*            queries against the KB and add instances for        */
/*            simple attacks based on results from the queries    */
/*            to the attacks ontology in the KB.               */
/* API: this program uses the Jena ontology API.                  */
/*                                              */
/*****************************************************************/


// imports for Jena API
import com.hp.hpl.jena.update.GraphStore;
import com.hp.hpl.jena.update.GraphStoreFactory;
import com.hp.hpl.jena.update.UpdateAction;
import com.hp.hpl.jena.update.UpdateFactory;
import com.hp.hpl.jena.update.UpdateProcessor;
import com.hp.hpl.jena.update.UpdateRequest;
import com.hp.hpl.jena.update.UpdateExecutionFactory;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.Syntax;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;

// general imports
import java.lang.*;
import java.io.*;
import java.util.*;
```

```
public class SimpleAttacks {
  /*****************************************************************/
  /*****                                              *****/
  /*****                  Add Land Instances                       *****/
  /*****                                              *****/
  /*****************************************************************/
  public static double addLandAttacks(PrintStream outputFile,
                              double addAttacksTime) {

   try {

     System.out.println("\tAdding Land instances...");

     // insert triples for Land attacks (TCPPacket with DIP=SIP and
     // Dest port = Src port).
    String queryStr =
     "PREFIX attack: " +
     "<" + KButility.URL_PREFIX + "attack.owl#> " +
     "PREFIX traffic: " +
     "<" + KButility.URL_PREFIX + "traffic.owl#> " +
     "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
     "INSERT " +
     "{ " +
     "   ?attack rdf:type attack:Land; " +
     "         attack:beginDate ?dateTime; " +
     "         attack:endDate ?dateTime; " +
     "         attack:description \"Land attack\"; " +
     "         attack:hasTargetIP ?destIP .  " +
     "} " +
     "WHERE  { { " +
     "   SELECT ?packet ?dateTime ?destIP " +
     "   { " +
     "      ?packet rdf:type traffic:TCPPacket; " +
     "            traffic:dateTime ?dateTime; " +
     "                        traffic:hasDestIP ?destIP; " +
     "                        traffic:hasSrcIP ?srcIP; " +
     "                        traffic:l4DestPort ?l4DestPort; " +
     "                        traffic:l4SrcPort ?l4SrcPort . " +
     "     FILTER ( ( ?destIP = ?srcIP ) && " +
     "            ( ?l4DestPort = ?l4SrcPort ) ) . " +
     "   } " +
     " } " +
     "LET (?attack := ?packet) . " +
     "}";
```

```
       addAttacksTime = addAttacksTime + KButility.execUpdQuery(queryStr, outputFile,
false);

      outputFile.println("Added Land Attacks!");
      outputFile.println();

   }  // end initial try

   catch(Exception e) {
          e.printStackTrace();
   }  // end catch

  return addAttacksTime;
  }   // end function addLandAttacks


  /*******************************************************************/
  /*****                                                      *****/
  /*****                Add ARP Spoof Instances                     *****/
  /*****                                                      *****/
  /*******************************************************************/

  public static double addARPSpoofAttacks(PrintStream outputFile,
                                  double addAttacksTime) {

  try {

   System.out.println("\tAdding ARP Spoof instances...");

   // same MAC, two different IP addresses
   String queryStr =
      "PREFIX attack: " +
      "<" + KButility.URL_PREFIX + "attack.owl#> " +
      "PREFIX traffic: " +
      "<" + KButility.URL_PREFIX + "traffic.owl#> " +
      "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
      "INSERT " +
      "{ " +
      "   _:a rdf:type attack:ARPSpoof; " +
      "      attack:beginDate ?startDateTime; " +
      "      attack:endDate ?endDateTime; " +
      "      attack:description \"ARP Spoof attack\"; " +
      "      attack:hasTargetMAC ?targetMAC . " +
      "} " +
      "WHERE { { " +
    "   SELECT ?targetMAC " +
    "        (MIN(?startTime) as ?startDateTime) " +
    "        (MAX (?endTime) as ?endDateTime) " +
    "        (count(?targetMAC) as ?cnt) " +
    "   { " +
    "    SELECT DISTINCT ?packet1 ?startTime ?endTime ?targetIP ?targetMAC " +
    "    { " +
    "     ?packet1 rdf:type traffic:L3Stream; " +
    "            traffic:startTime ?startTime; " +
    "            traffic:endTime ?endTime; " +
    "            traffic:hasNode1MAC ?targetMAC; " +
    "            traffic:hasNode1IP ?targetIP . " +
    "     { " +
    "   SELECT DISTINCT ?packet2 ?startTime2 ?endTime2 ?targetIP2 ?targetMAC "+
    "       { " +
    "         ?packet2 rdf:type traffic:L3Stream; " +
    "               traffic:startTime ?startTime2; " +
    "               traffic:endTime ?endTime2; " +
    "               traffic:hasNode1MAC ?targetMAC; " +
    "               traffic:hasNode1IP ?targetIP2 . " +
    "       } " +
    "     } " +
    "    FILTER ( ( ?packet1 != ?packet2 ) &&  " +
    "          ( ?targetIP != ?targetIP2 ) ) . " +
    "    } " +
    "   } " +
    "   GROUP BY ?targetMAC " +
    " } " +
    "}";

   addAttacksTime = addAttacksTime + KButility.execUpdQuery(queryStr, outputFile,
false);

  // same IP, two different MAC addresses
  queryStr =
    "PREFIX attack: " +
    "<" + KButility.URL_PREFIX + "attack.owl#> " +
    "PREFIX traffic: " +
    "<" + KButility.URL_PREFIX + "traffic.owl#> " +
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
    "INSERT " +
    "{ " +
    "   _:a rdf:type attack:ARPSpoof; " +
    "      attack:beginDate ?startDateTime; " +
    "      attack:endDate ?endDateTime; " +
    "      attack:description \"ARP Spoof attack\"; " +
    "      attack:hasTargetIP ?targetIP . " +
    "} " +
    "WHERE { { " +
    "   SELECT ?targetIP " +
    "          (MIN(?startTime) as ?startDateTime) " +
    "          (MAX (?endTime) as ?endDateTime) " +
    "          (count(?targetIP) as ?cnt) " +
    "   { " +
    "     SELECT DISTINCT ?packet1 ?startTime ?endTime ?targetIP ?targetMAC " +
    "     { " +
    "       ?packet1 rdf:type traffic:L3Stream; " +
    "                traffic:startTime ?startTime; " +
    "                traffic:endTime ?endTime; " +
    "                traffic:hasNode1MAC ?targetMAC; " +
    "                traffic:hasNode1IP ?targetIP . " +
    "      { " +
    "    SELECT DISTINCT ?packet2 ?startTime2 ?endTime2 ?targetIP ?targetMAC2 "+
    "        { " +
    "          ?packet2 rdf:type traffic:L3Stream; " +
    "                   traffic:startTime ?startTime2; " +
    "                   traffic:endTime ?endTime2; " +
    "                   traffic:hasNode1MAC ?targetMAC2; " +
    "                   traffic:hasNode1IP ?targetIP . " +
    "        } " +
    "      } " +
    "     FILTER ( ( ?packet1 != ?packet2 ) &&  " +
    "            ( ?targetMAC != ?targetMAC2 ) ) . " +
    "     } " +
    "   } " +
    "   GROUP BY ?targetIP " +
    " } " +
    "}";

   addAttacksTime = addAttacksTime + KButility.execUpdQuery(queryStr, outputFile,
false);


     outputFile.println("Added ARP Spoof Attacks!");
     outputFile.println();

  }  // end initial try

  catch(Exception e) {
         e.printStackTrace();
```

```
 }  // end catch

return addAttacksTime;
}   // end function addARPSpoofAttacks



/******************************************************************/
/*****                                                      *****/
/*****       Add addTCPConnect PacketColletion Instances            *****/
/*****                                                      *****/
/******************************************************************/
public static double addTCPConnect(PrintStream outputFile,
                              double addAttacksTime) {

try {

        System.out.println("\tAdding TCP Connect instances...");
        outputFile.println("Adding TCPConnect instances from TCPPacket...");

        // Retrieve all TCP Packet instances to same destIP and tcpRstFlag is true
        // It is important to note that the TCP connect attack source and
        //   destination addresses are reversed from the TCPPacket instance
        //   to the TCPConnect attack instance. This is due to the fact that
        //   the TCP connect attack is identified with the RST flag set in
        //   the TCP packet, which is actually done in the response to the
        //   SYN packet, which is sent from the attacker. So, the destination
        //   address in the RST packet is actually the attacker (source of
        //   the attack).
 String queryStr =
   "PREFIX traffic: " +
   "<" + KButility.URL_PREFIX + "traffic.owl#> " +
   "PREFIX attack: " +
   "<" + KButility.URL_PREFIX + "attack.owl#> " +
   "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
   "INSERT  { " +
   "   _:a rdf:type attack:TCPConnect; " +
   "      attack:beginDate ?beginDateTime; " +
   "      attack:endDate ?endDateTime; " +
   "      attack:description \"TCP Connect attack, predict TCP Sequence Number\"; " +
                              "      attack:hasTargetIP ?srcIP; " +
                              "      attack:scanFrequency ?cnt . " +
                              "} " +
   "WHERE { { " +
      "   SELECT ?srcIP " +
      "         (MIN(?dateTime) as ?beginDateTime) " +
      "         (MAX(?dateTime) as ?endDateTime) " +
      "         (count(?srcIP) as ?cnt) " +
      "   { " +
      "    SELECT DISTINCT ?packet1 ?destIP ?srcIP ?dateTime " +
      "     { " +
      "      ?packet1 rdf:type traffic:TCPPacket; " +
      "              traffic:dateTime ?dateTime; " +
      "              traffic:hasDestIP ?destIP; " +
      "              traffic:hasSrcIP ?srcIP; " +
      "                         traffic:tcpRstFlag true . " +
      "     { " +
      "        SELECT DISTINCT ?packet2 ?destIP ?srcIP ?dateTime2 " +
      "        { " +
      "          ?packet2 rdf:type traffic:TCPPacket; " +
      "                  traffic:dateTime ?dateTime2; " +
      "                  traffic:hasDestIP ?destIP; " +
      "                  traffic:hasSrcIP ?srcIP; " +
      "                             traffic:tcpRstFlag true . " +
      "        } " +
      "     } " +
      "     FILTER ( ?packet1 != ?packet2 ) . " +
      "   } } " +
      "  GROUP BY ?srcIP " +
      "  HAVING (count(?srcIP) > 0) " +
      " } " +
      "}";

    addAttacksTime = addAttacksTime +
                       KButility.execUpdQuery(queryStr, outputFile, false);

   }  // end initial try

   catch(Exception e) {
          e.printStackTrace();
   }  // end catch

  return addAttacksTime;
  }   // end function addTCPConnect


}   // end class SimpleAttacks
```

Vita

Lisa Frye received two undergraduate degrees in 1990 (Bachelor of Science, Computer Information Science and Bachelor of Science in Education, Mathematics) and a Master of Science degree in 1993 (Computer Information Science) from Kutztown University, Kutztown, PA. For four years she worked as a Support Specialist at Unisys Corporation and an Application Developer at EDS. She returned to her Alma Mater as a professional employee in several positions within the Information Technology department of the university, including System Administrator and Server Manager. With a background in Education and a Master's degree in Computer Science, Ms. Frye was recruited in 1997 as an adjunct professor for Reading Community College, Harrisburg Area Community College and Muhlenberg College to teach computer science and programming; she taught in this capacity for four years while maintaining fulltime employment with the Information Technology Department at Kutztown. In 2001, Ms. Frye made a full time shift from industry to the classroom when she accepted a full time position as a computer science faculty member. For the last ten years, she has been a faculty member in the Computer Science Department at Kutztown University; where she currently is an Associate Professor. Ms. Frye has been pursuing her Ph.D. at Lehigh University, Bethlehem, PA under the guidance of Professor Liang Cheng in the Laboratory Of Networking Group (LONGLAB). Her research focuses on using ontology to aid in the network management of heterogeneous multi-tier networks.