IDENTIFYING APPLICATION LEVEL PROTOCOLS BY ANALYZING COMMUNICATION PATTERNS OVER MULTIPLE PORTS

BY

LEE BAILEY

A Thesis Submitted to the Graduate Faculty of WAKE FOREST UNIVERSITY GRADUATE SCHOOL OF ARTS AND SCIENCES in Partial Fulfillment of the Requirements for the Degree of MASTER OF SCIENCE Computer Science May, 2012 Winston-Salem, North Carolina

Approved By:
Errin Fulp, Ph.D., Advisor
William Turkett, Ph.D., Chair
Daniel Canas, Ph.D.
Acknowledgments

There are many people who helped to make this thesis possible. I’d like to thank professors Errin Fulp and William Turkett for both guiding me in my research and assisting me in the difficult process of writing my first thesis. I’d also like to give credit where it’s due to my fellow Wake Forest graduate student colleagues Eddie Allan and Chaz Lever. Eddie provided the theoretical foundation of motif-based application identification upon which this thesis extends to a new domain, multi-port applications. Chaz is the master-mind behind much of the brilliantly engineered toolchain implementation of Eddie’s theoretical work. Finally, I’d like to thank Timothy Thomas for helping with generating some of our datasets. Without help from these people, this thesis would never have been possible.
## Table of Contents

**Acknowledgments** ................................................................. ii

**List of Figures** ............................................................................. v

**Abstract** ...................................................................................... vi

**Chapter 1 Introduction** ............................................................... 1

1.1 Need for Network Management and Security .......................... 2
1.2 Difficulty of Identifying Network Applications ..................... 3
1.3 Multi-port Graph Analysis ......................................................... 4
    1.3.1 New Approach .............................................................. 5

**Chapter 2 Basics of Packet-Switched Networks** ...................... 7

2.1 Computer Networks ................................................................. 7
2.2 TCP/IP Networks and Protocols ............................................. 7
    2.2.1 Transport-Layer Ports ................................................. 9
2.3 Network Flow Data ................................................................. 12

**Chapter 3 Techniques for Identifying Network Applications** ...... 14

3.1 Importance of Application Identification ............................... 14
3.2 Port-Based Identification ....................................................... 15
3.3 Signature-Based Identification .............................................. 15
3.4 Statistics-Based Classification .............................................. 17
    3.4.1 Classification Using Packet Sequences ....................... 17
3.5 Connection-Based Identification ........................................... 18

**Chapter 4 Communication Graph Analysis Techniques** ............ 20

4.1 Graph Types and Measures .................................................... 20
    4.1.1 Types of Graphs ..................................................... 21
    4.1.2 Graph Motifs ....................................................... 22
    4.1.3 Traditional Graph Measures .................................... 23
4.2 Graph Based Application Identification ................................ 24
    4.2.1 BLINC ................................................................. 25
    4.2.2 Graption ............................................................. 26
4.3 Motif Approach to Application Classification ........................ 29
Chapter 5  Multiple-Port Graph Analysis for Application Identification ................................................................. 30
  5.1  Applications and the Use of Multiple Ports ................................. 30
  5.2  Using Motifs for Application Identification ............................... 32
      5.2.1  Data Collection .................................................. 32
      5.2.2  Application Graphs .............................................. 33
      5.2.3  Vertex Profiles .................................................. 34
  5.3  Implementation of Motif-Based Application Identification ............ 36

Chapter 6  Experimental Results ......................................................... 40
  6.1  Network Traces ............................................................ 40
      6.1.1  Sources of Traces ............................................... 41
  6.2  Experimental Set-up ....................................................... 44
      6.2.1  Metrics Used to Measure Performance ......................... 45
  6.3  Results and Analysis ..................................................... 46

Chapter 7  Conclusions and Future Work ........................................... 51
  7.1  Network Application Identification ...................................... 51
  7.2  Motif-Based Application Identification using Multiple Ports ........ 52
  7.3  Areas for Future Work ................................................... 53
      7.3.1  Using Other Ports ............................................... 53
      7.3.2  Tor and Onion Routing Issues .................................. 54

Bibliography .............................................................................. 55

Curriculum Vitae ........................................................................ 59
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A Diagram of the Full IP Suite [27]</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Details of the IP Suite with Layers of Abstraction [27]</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Examples of TCP/IP Protocols at each Layer [27]</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>The Difference between Connection-oriented and Connectionless Communication</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Theoretical Packet Structure with Header and Payload Division</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Web application example</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>A graph with 6 vertices and 6 edges</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>A directed graph with 6 vertices and 6 edges</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>Examples of various size 3 motifs</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>Visual representations of transport-layer interactions for various applica-</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>tions: port numbers are provided for completeness but are not used in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>classification [21]</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>The Gnutella Traffic Dispersion Graph (P2P application) [19]</td>
<td>27</td>
</tr>
<tr>
<td>4.6</td>
<td>The HTTPS Traffic Dispersion Graph (client-server application) [19]</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>The workflow used for associating motif profiles with applications.</td>
<td>32</td>
</tr>
<tr>
<td>5.2</td>
<td>At left is an example application graph that consists of 15 nodes, while</td>
<td></td>
</tr>
<tr>
<td></td>
<td>at right is a motif of size 3 that occurs 5 times in the application graph.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Node 6, involved in the size 3 motif with nodes 5 and 10, also interacts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>separately with node 9. [9]</td>
<td>33</td>
</tr>
<tr>
<td>5.3</td>
<td>A List of n Vertices in a Vertex Profile [16]</td>
<td>35</td>
</tr>
<tr>
<td>5.4</td>
<td>First section of the toolchain</td>
<td>37</td>
</tr>
<tr>
<td>5.5</td>
<td>Second section of the toolchain</td>
<td>38</td>
</tr>
<tr>
<td>5.6</td>
<td>Third section of the toolchain</td>
<td>39</td>
</tr>
</tbody>
</table>
Abstract

Lee Bailey

Discovering the specific applications running on a computer network is of utmost importance for quality of service and other resource management, network planning, network security, and usage policy enforcement. Unfortunately, the evolution of application protocols has rendered traditional network application methods such as transport layer port-based identification and deep packet inspection nearly useless as applications tend to use non-standard ports and traffic is increasingly encrypted.

Recently developed connection-based application discovery techniques can identify network applications using only high-level communication information. For example, motif-based techniques classify network applications by identifying communication sub-graphs (motifs). This thesis seeks to improve the accuracy of motif-based approaches by considering graphs built from the multiple services (ports) an application may use. Using the additional information provided by multiple ports, this thesis may be able to improve network application classification accuracy.
Chapter 1: Introduction

Monitoring network traffic and classifying applications are essential tasks for network administrators. The importance of these tasks spans various network management activities including network security, usage policy enforcement, and network resource allocation. For example, detecting P2P traffic is of particular importance to ISPs under immense pressure to manage such traffic by groups such as the entertainment industry in legal and copyright disputes. From a network security standpoint, as systems grow in size and applications grow in complexity, the need to monitor and fully understand all traffic flow becomes increasingly difficult.

Unfortunately, the evolution of application protocols has rendered traditional network application methods such as transport layer port-based identification nearly useless. This thesis proposes a new approach relying on motif graphs modeling multiple port applications’ interactions from a high-level behavioral vantage point. The method is “blind” in that it does not inspect packet headers or packet payload making it both harder to circumvent and protective of privacy concerns. It is more robust to evasion techniques than traditional methods because an application’s behavior is much more difficult to change compared to its transport-layer port number and other low-level arbitrary usage features.

This thesis extends the work of fellow Wake Forest colleague, Eddie Allan. Allan’s approach pioneered the use of motif-based graphs, showing that using motifs resulted in a significant increase in classification performance compared to traditional graph-based identification methods [16]. The approach only classified single-port protocols. It essentially bound protocols (e.g. AOL Instant Messenger, HTTP, DNS, Kazaa, Microsoft Active Directory (MSDS), NetBIOS Name Service, and Secure Shell (SSH))
to a single port and assumed that any traffic on these ports was associated with that specific protocol.

In reality, many different applications use multiple ports as part of their overall functionality. In particular, every application studied in this research uses both HTTP and DNS for its operation. Therefore it is much more useful and practical to be able to classify on higher-level applications using information collected over multiple ports.

1.1 Need for Network Management and Security

Discovering the specific applications running on a computer network is of utmost importance for quality of service and other resource management, network planning, network security, and usage policy enforcement. For example, it is important to determine the resource-intensive applications currently running such as video-conferencing in order to provide sufficient bandwidth to support such applications. Additionally, knowing the full range of applications using the network can assist with detecting security breeches such as botnet activities. This knowledge can also enforce usage policy by effectively banning applications, for example software used for multimedia piracy. These are just a few of the countless number of examples of the need to fully understand the types of applications that can provide network administrators detailed knowledge of the all of the specific applications currently utilizing the network’s resources.

A higher-level understanding of the current usage of the network can also significantly improve how networks are secured and can assist network management techniques that specify policy by function and usage rather than packet-level information that is common for traditional firewalls [9]. Most organizations an Acceptable Usage Policy (AUP) to protect themselves from litigation and to ensure the security of the network. The SANS security project provides several examples of AUPs for
various organizations [6]. An AUP is a set of rules that restrict the ways that a network site or system may be used. For example, a company may prohibit its employees from playing certain games over their network, or a university may prohibit hosting a web server on its network. Such behavior can lead to consequences ranging from a verbal/written warning to termination of employment/expulsion from a university or even prosecution under a court of law depending on the severity of the offense. In reality, most AUPs are very lightly enforced, largely due to the difficulty of mapping high-level descriptions of rules to traditional application identification by traditional port-based methods [2]. In any case, it is hoped that the process of identifying such breeches of the AUP will be greatly improved by the method proposed in this thesis. Being able to overview the usage policy and the currently running applications on the same plane is the key.

1.2 Difficulty of Identifying Network Applications

A network application is any application that communicates across a computer network. Originally, the task of identifying network applications was fairly simple as most applications used specific well-known ports discussed previously. The process was simple; all traffic on a particular port was assumed to be part of the application that is assigned to that port in the Internet Assigned Numbers Authority (IANA). However, over time network applications have become more complex and identifying them in the traffic mix has become much more challenging for a variety of reasons.

The sheer number of network applications in existence has grown astronomically, and it is not easy to maintain a simple mapping of applications to well-known port numbers [20]. Moreover, many applications use dynamically-allocated ports “on the fly” or user-chosen ports, so that maintaining such a list is rendered useless. It is becoming more common for applications to be tunneled through port 80 so that
HTTP traffic actually represents parts of many different applications. Additionally traffic is increasingly encrypted rendering packet-inspection methods useless.

To address these growing concerns, new methods of network application identification are being developed. These methods can be broadly grouped into a few categories: port-based identification, signature-based identification, statistics-based identification, and graph-based identification. This thesis proposes a hybrid that uses properties of statistics-based and graph-based identification and overcomes the shortcomings of the first two methods.

1.3 Multi-port Graph Analysis

This thesis focuses on developing a multi-port connection analysis technique to better identify multi-port applications which are simply applications that use more than one transport layer port for communication. Often this takes the form of dependencies on different lower-level services for the application to operate reliably and efficiently [12]. Allan previously attempted to identify multi-port applications using only a single port in his analysis, and this thesis proposes extending his approach to consider multiple ports simultaneously in the classification process [16]. Examples of these high-level applications include web, email, instant messaging, multiplayer online computer games, file-sharing, and online audio/video conferencing. Each of these operates on a distributed set of clients and servers, and each depends on several supporting services such as Active Directory (AD), Domain Name System (DNS), Kerberos, and Windows Internet Name Service (WINS) [12].

For example, a simple webpage requested issued by an end-user in a browser relies on DNS to resolve the host IP address, HTTP to send the GET request, and most likely a database call to a server. An online computer game most likely uses many more services. For example, consider the Massively Multiplayer Online Roleplaying
Game (MMORPG), World of Warcraft (WoW), played by over 10 million people on hundreds of different servers all over the world. WoW uses TCP ports 3724, 1119, 1120, 4000, 6112, 6113, 6114, and 6881 - 6999 for game logic and patching as well as UDP ports 1119 and 3724 for Voice Chat [1].

Given there is less reliance on a single port for communication, application discovery methods should consider the communication over multiple ports. In order to better classify these applications a new approach needs to be devised.

1.3.1 New Approach

The approach developed in this thesis to identify multi-port applications draws from both statistics-based and graph-based identification methods. First an application graph is constructed for each relevant port to each application. These graphs are then mined to discover the significant motifs, that are patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks [24]. Next these individual port graphs are combined to create a mega-graph for each application of all the ports that are considered to be relevant to defining its overarching behavior. Finally the motif data is sent to the Support Vector Machine (SVM) for classification.

Objectives and Advantages

The main objective of this thesis is to extend the prior work done by Wake Forest Alumni Eddie Allan and Chaz Lever, and professors Dr. Errin Fulp and Dr. William Turkett to be able to classify a wider range of higher-level applications, multi-port applications. Joining the project a couple of years after it was initiated enables standing on the shoulders of a plethora of hard work performed at the low-level to build the toolchain and construct the theoretical approach. In a sense, this thesis will
present a modification to a past approach which will open the uses to a much more practical group of applications.

Allan first constructed the process of using motifs to classify applications using a single port. He was able to achieve an average of around 85% accuracy rate compared to a 79% accuracy rate he observed using only traditional graph measures to identify the same protocols. Lever modified Allan’s approach to include more metrics than merely node type, the presence or absence of certain size 3 or size 4 motifs. He considered a multitude of node and edge features such as duration, bytes sent, packets sent, number of connections, etc. Using this additional information, he was able to increase the accuracy rate to 94% over the applications studied in this thesis.

Building on the past motif-based work of the project, this thesis aims to use information from multiple ports to achieve higher classification accuracy on both low-level protocols and higher-level applications. Although past work has shown single-port analysis to be reasonably viable, the results show that considering a few additional ports in the analysis stage decreases uncertainty and increases accuracy significantly. However, accuracy is not the only consideration; this thesis hopes to achieve higher accuracy without sacrificing speed of identification and maintaining resilience to the deceptive tactics of intruders by modeling application communication at a high-level that is nontrivially associated with intrinsic behavior. The technique should have all the advantages of the previous motif-based method, yet achieve a higher identification accuracy.
To understand the remainder of the paper and the goal at hand, it is necessary to review the basics of packet-switched networks and network applications. Specifically this chapter will review the TCP/IP networks and the associated protocols.

2.1 Computer Networks

A computer network is a collection of hardware components and computers interconnected by some communication channels that enables the sharing of resources and information among the participating computers. If at least one process in some computer is able to send/receive data to/from at least one process in a different computer, then the two computers are said to be connected in a computer network [27]. The figure below provides a visual model of a computer network with lines drawn between devices representing some type of communication channel. A Packet-switched computer network is one in which all transmitted data, regardless of content, type or structure is grouped into suitably sized blocks called packets. Each packet is routed independently, and transmission resources are allocated as needed in order to optimize utilization of available link capacity, minimize response times, and increase the robustness of the communication. The transmission of packets proceeds with variable delay and throughput depending on the local traffic load in the network [27].

2.2 TCP/IP Networks and Protocols

The Internet protocol suite is the set of communication protocols used to manage communication for the Internet and other similar computer networks. It is also known as TCP/IP because of its two most important protocols, Transmission Control Pro-
protocol (TCP) and Internet Protocol (IP). TCP provides connection-based, reliable and ordered delivery of bytes from a process on one computer to a process on a different computer [27]. It is the protocol that major applications such as the World Wide Web, email, and file transfer rely on. Other applications which do not require connection-oriented or a reliable data stream service may choose to instead use the User Datagram Protocol (UDP) which emphasizes reduced latency over reliability.

Figure 2.1: A Diagram of the Full IP Suite [27]

The Internet protocol suite contains four abstraction layers:

1. The **link layer** containing communication technologies for a local network.

2. The **internet layer** connecting local networks and forming the Internet.

3. The **transport layer** handling host-to-host communication.

4. The **application layer** handling application-based interaction on a process-to-process level between communication internet hosts.

It will be shown that approaches for identifying the application will rely on information that resides at a certain layer. For example, the technique described in this
thesis relies on the communication interactions that occur at the transport and application layers but is “blind” to the actual data transmitted at the lower layers. This higher-level approach is advantageous as it alleviates privacy concerns and evades the problem of encrypted data.

### 2.2.1 Transport-Layer Ports

Before discussing the low-level details of computer network interaction, it is important to discuss the higher-level picture of what’s actually occurring between two intercom-
municating processes. In order to connect to each other, applications create a *session*, a mutually agreed-upon information exchange. A session is the basic requirement of both connection-oriented communication and connectionless communication modes. Figure 1.4 illustrates the difference between connection-oriented and connectionless communication. Application layer examples of session implementation include HTTP and telnet remote login.

<table>
<thead>
<tr>
<th>Service</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable message stream</td>
<td>Sequence of pages</td>
</tr>
<tr>
<td>Reliable byte stream</td>
<td>Remote login</td>
</tr>
<tr>
<td>Unreliable connection</td>
<td>Digitized voice</td>
</tr>
<tr>
<td>Unreliable datagram</td>
<td>Electronic junk mail</td>
</tr>
<tr>
<td>Acknowledged datagram</td>
<td>Registered mail</td>
</tr>
<tr>
<td>Request-reply</td>
<td>Database query</td>
</tr>
</tbody>
</table>

Figure 2.4: The Difference between Connection-oriented and Connectionless Communication

The *transport layer* provides host-to-host communication services for applications through either TCP or UDP [5]. It provides services such as connection-oriented communication, reliability, flow control, congestion avoidance, and multiplexing. The transport layer’s main responsibility is the delivery of data to the appropriate application process on the host computers. This process involves creating data packets and adding source and destination *port* numbers in the header of each packet. Combining these port numbers with the source and destination IP address of layer 3 produces the *network socket*, an identification address of the process-to-process communication [27].

A *port* is an application-specific or process-specific software construct serving as a communication’s endpoint for a particular host’s operating system. Note that the concept of the port is entirely non-physical. A port is associated with the IP address of the host as well as the type of protocol used for communication, such as TCP or
UDP. For the TCP/IP protocol suite, a port is labeled as a 16-bit number, commonly referred to as the *port-number*. The port number is the final destination address in the host-to-host communication process [27]. Different hosts and different protocols can both use the same port number for communication.

Many ports are reserved by specific protocols. For example, TCP reserves port numbers 0-1024 for "well-known" ports used to access services that are standardized across the internet [5]. The rest of the ports can be freely allocated to application processes. The following are a few examples of well-known ports:

- 20, 21: File Transfer Protocol (FTP)
- 22: Secure Shell (SSH)
- 23: Telnet
- 25: Simple Mail Transfer Protocol (SMTP)
- 53: Domain Name Service (DNS)
- 80: Hypertext Transfer Protocol (HTTP)
- 443: HTTP Secure (HTTPS)

Ports 53 and 80, DNS and HTTP respectively, are of particular interest to this thesis because all of the multi-port applications studied involve some traffic across both of them. Port 80 is by far the most used port because many applications are tunneled through HTTP (using the port as it was not intended to be used) for some reason, possibly to evade traditional port-based firewalls.
2.3 Network Flow Data

This thesis uses Cisco NetFlow Records to monitor and analyze network traffic [3]. NetFlow is a network protocol for collecting IP traffic statistics on all interfaces on which NetFlow is enabled. These statistics are later exported as NetFlow records. Cisco defines network flow as a unidirectional sequence of packets that all share at least the following characteristics [3]:

1. Ingress interface (SNMP)
2. Source IP address
3. Destination IP address
4. IP protocol
5. Source port for UDP or TCP, 0 for other protocols
6. Destination port for UDP or TCP, type and code for ICMP, or 0 for other protocols
7. IP Type of Service
8. Bytes sent
9. Packets sent
10. Duration

Given the amount and speed of data transmitted in modern networks, it is no longer feasible to capture full packet data from management purposes. Flowdata was created to allow some record of data transmissions with minimal space requirements. For the purpose of this thesis, NetFlow provides more than enough information to
gather statistics on the applications’ connections relevant to identifying each’s behavior.

Three different views of the same network:

The above diagram illustrates viewing a network with three different levels of abstraction. The physical network is layer 2 in the IP suite. It is the physical topology of the network, the view of how the network transfers data at the lowest level. The logical view is layer 3 in the IP suite. It represents how packets are routed around a computer network logically. Finally, the application network is the highest level of abstraction in which the network is viewed as host-to-host interaction. Only source and destination are important in this view, as the physical and logical details of how the packets arrive from source to destination are abstracted away. This thesis takes a high-level approach and is only interested in the application view of a network.
Chapter 3: Techniques for Identifying Network Applications

Over the last two decades, many different techniques have been developed in an attempt to accurately identify network applications. These include but are not limited to port-based techniques, signature-based methods, statistics based classification, and connection pattern based methods. Though each of these methods contain certain strengths and weaknesses, they all fall short of the high standard of accuracy, speed, resilience, and robustness demanded by the ever-evolving nature of computer software and networks.

3.1 Importance of Application Identification

Identifying the applications in a network traffic mixture is important to assist network administrators with various network management activities including securing the network from intruders, enforcing usage policy by preventing illegal programs from running, identifying P2P traffic of potentially copyright-sensitive data, designing network equipment [26], and properly allocating network resources among other tasks. For instance, it is important to determine if a user is participating in a resource-intensive video conference so that proper bandwidth can be allocated. Moreover, in terms of security, it is important to properly identify bonnet activity and software piracy to take appropriate management and legal actions. Identifying the actual network applications in the traffic mix provides a higher-level management interface than simple firewalls which only monitor ports and thus have no real knowledge of the current applications. Such an application-level awareness will assist in keeping networks secure from intruders, efficient in proper resource allocation and immune
3.2 Port-Based Identification

The first and simplest method developed to identify network applications is port-based identification. This method identifies the application by observing its transport layer port number, and it only needs access to the header of the packets. It relies on the principle assumption that a port number will uniquely identify an application type. Therefore, to manage the access of a particular application, an administrator need only to add or remove a rule to/from the firewall permitting or blocking any packets with a certain port number.

Originally this method was fast, efficient, and accurate in the past when most services used well-known ports [26]. However, today the port-based technique has become much less accurate for a plethora of reasons. Many new applications, especially P2P applications, do not utilize well-known ports. Some even use random ports or user-defined ports instead of a default. Another reason is that many modern applications are tunneled through port 80 to circumvent common firewalls that block some unauthorized or unknown applications. Therefore existing port-based classification methods will often misclassify tunneled applications as web traffic [18].

3.3 Signature-Based Identification

In signature-based identification, the identification is achieved by using pre-defined byte signatures of specific applications in the packet payload. As an example, consider the 'GET' request of the HTTP protocol. A web page is retrieved by sending a 'GET' request of a particular format to a web server. Knowing this information ahead of time, a rule could be constructed matching all packets that contain such an aptly
formatted request to web traffic.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>5</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Header</td>
<td>User data</td>
</tr>
</tbody>
</table>

Figure 3.1: Theoretical Packet Structure with Header and Payload Division

This method is also called packet-based identification because it requires access to both the packet header and packet payload. Figure 3.1 illustrates the difference in the packet header and packet payload or user data. The method searches the packets for the signature patterns defined in some database. For example Layer-7 Filter (L7) is an open source filter that seeks to determine the application based on string patterns that occur in the packet payloads. For example signature for L7 HTTP packets are stored by the system, reassembled into the original data stream then searched for patterns. For example, if L7 finds HTTP header information in the payload, it assumes the flow is HTTP. Of course, it is possible that a legitimate FTP session could transmit the same patterns, and be incorrectly classified as HTTP. Along with port-based identification, signature-based identification is a classical traffic classification method [26].

Obvious downsides to this method include its reliance on the accuracy of the signature database, its resource-intensive requirements, and its reliance on unencrypted data among others. Attackers can easily evade this method by encrypting their traffic, or disguising their traffic as legitimate by inserting fragments of other signatures. In order to be useful, the signatures must be updated frequently or else many applications will be misclassified. For example the performance of L7 is such that it is not recommended for security purposes due to the high number of miss-classifications (false positives) [13]. Since this method involves inspecting the detailed contents of network traffic, privacy issues rise as another concern. Finally, since classifiers can only use a finite amount of packets per connection, the inspected portion may not
3.4 Statistics-Based Classification

Statistics-based classification involves analyzing and identifying the traffic by some payload-independent statistical feature. These statistical features include the minimum, maximum, mean, and standard deviations of forward/backward packet length and inter-arrival packet time; protocol; duration of the flow; and number of packets/bytes in forward/backward direction, among others [29]. The process involves identifying which features are significant in a session and classifying the application though some artificial intelligence or machine learning technique.

The machine learning techniques include Bayesian Analysis [25], Support Vector Machines (SVM) [31], Decision Trees [23], and Nearest Neighbor [22], etc. Usually hand-classified data is used for training and testing data sets. Gathering this ground truth data is non-trivial and time-intensive. This is the main weakness in this method; a lot of previously accurately hand-classified data is needed to classify the rest of the trace [26].

3.4.1 Classification Using Packet Sequences

As a further example of the use of packet sequences to classify network traffic, consider the previous work completed by fellow Wake Forest computer science graduate student, Andrew Karode. Karode addressed a similar problem to the one identified in this thesis, single-flow in-the-dark TCP traffic classification. Recall that in-the-dark is a constraint that allows limited inspection of the packet header and no information of the packet payload. He examined only features that describe the session flow: packet size, direction and inter-arrival time [28].

Karode employed multiple one-versus-all binary SVMs to deal with the many
different protocols he sought to classify. The classifiers were trained on known flows for each particular protocol as positive examples and all other protocols as negative examples. The data was then tested by serving as input to each individual classifier successively and classified as the protocol for which classifier it received the highest score [28]. The key insight is his method is able to identify the application based on the packet sequence observed in the communication of two computers. In the end, Karode’s work illustrates the usefulness of statistical based identification but still requires packet statistics that may not be available.

3.5 Connection-Based Identification

In connection-based identification, the basic idea is to examine the communication pattern generated by a particular host and compare it to different applications. This is a high-level approach that does not examine the contents of packet headers, packet payloads, or detailed packet statistics. The communication between hosts in a network, not the physical topology of the network, define the network graph of interest to this method.

As an example, consider the typical connections created by a modern web application. When a user requests a page through a web browser, the web browser must first resolve the URL provided by the user to a physical IP address through the DNS. Then the web server contacts a database with the user’s query parameters. These
parameters could be the time of day and the location of the user. The database then processes this information and sends it back to the web server. The web server then builds the html page and sends the output back to the user’s browser to display the page. This request generated six distinct communications. However, communications labeled 3 and 6 occur within a single machine and would not be observable from the outside. These computer-to-computer connections are modeled into an application graph. As seen in figure 3.2, this web-page request can potentially generate many distinct communications (connections).

The application graph features can provide insight into the activity of a network and can potentially assist with application identification [9]. For instance, these connection patterns describe network flow characteristics corresponding to different applications through capturing the relationships between the use of source and destination ports [26].

The approach of this thesis modifies the traditional connection-based identification method to search for sets of small subgraphs (motifs) associated with different network applications. The motifs are discovered by comparing relative occurrence of certain subgraphs in the total network application graph to randomized graphs of the same size. The shapes of motifs correspond to the differing applications, but some shapes are more represented among groups of applications than others.
Chapter 4: Communication Graph Analysis Techniques

The previous chapter reviewed various methods for identifying a network application. The last type of application identification method used connection-level information and for that reason is considered more robust. This chapter will review these connection oriented methods in greater detail.

Since communication patterns are the central information for these types of identification methods, it is important to review graphs and various graph metrics. This chapter first introduces the terminology of graphs and defines several different types. It then discusses motifs and their relation to application identification.

4.1 Graph Types and Measures

Originally taken from mathematics, graphs are collections of inter-connected objects capable of modeling systems in computer networking, biology, chemical compounds, social-groups, and communication systems, among others [11]. In computer science, a graph is an abstract data type consisting of a finite set of edges (ordered pairs) of certain vertices. Vertices, also known as nodes, are the fundamental units of a graph. For this thesis, they will represent host computers in a computer network graph. Vertices are linked together by edges which represent some relationship between the vertices such as physical links between computers in a computer network.

Formally, a graph $G$ consists of a finite, non-empty set of vertices $V$, connected by a set of edges $E$, written as $G = (V, E)$. The set of vertices $V$ is written $V = \{v_0, v_1, \ldots, v_k\}$. In figure 3.1, $V = \{1,2,3,4,5,6\}$. The cardinality of this set, $|V|$, is the order, or number of nodes in the graph. A graph’s edge set is defined as $E \subseteq u, vu, v \in V$. In figure 3.1, $E = \{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)\}$. For
brevity, an edge can be written $e_{ij}$ to mean an edge linking node $i$ to node $j$. $|E|$ is the number of edges in the graph, known as its size. The degree of a node, $\text{deg}(v)$, is the number of nodes that $v$ is adjacent to in the graph (those that can be reached by traversing one edge). This set of nodes is known as $N(v)$, the neighborhood of $v$. In Figure 3.1, nodes 2 and 3 are adjacent to node 1, and $N(1) = \{2, 3\}$ [16].

### 4.1.1 Types of Graphs

There are many different types of graphs that are useful in modeling complex systems. In an undirected graph, the edges are pairs of unordered vertices so that $e_{ij} = e_{ji}$. Figure 3.1 is an example of undirected graph. Undirected graphs are usually drawn with straight lines between the edges. On the other hand, the edges in a directed graph are ordered pairs of vertices, hence $e_{ij} \neq e_{ji}$. Figure 3.2 is an example of a directed graph. Unlike an undirected graph, in a directed graph, the order of the vertices in the pairs matters. Directed graphs are usually drawn with arrows indicating the direction between the edges. In figure 3.2, $V = \{A, B, C, D, E\}$, and $E = \{(A, B), (B, C), (D, C), (B, D), (D, B), (E, D), (B, E)\}$.

The degree measure, $\text{deg}(v)$, can be further specified as the indegree, $\text{id}(v)$, and outdegree, $\text{od}(v)$, of a vertex describing the number of vertices of $G$ from which $v$ is adjacent, and the number of vertices in $G$ to which $v$ is adjacent, respectively.
The associated undirected graph of a directed graph is obtained by disregarding the ordering of the end points of each edge; that is converting the directional arrows to straight lines [16].

As an example of an undirected graph, consider the information flow of a local computer network with multiple hosts connected together though ethernet. The host computers would be the vertices, and the edges would represent if the node is capable of sending to another. This graph is undirected because information can flow in both directions as each host can both send and receive data. Now consider adding a passive network monitor (sniffer) that is only interested in observing traffic in the network. The sniffer can only receive data. Therefore, this transforms the graph into an undirected graph as not every vertex can send and receive data.

### 4.1.2 Graph Motifs

Given a graph, it may consist of subgraphs. If these subgraphs occur in high numbers in the graph, then they are referred to as motifs. Formally, motifs are patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks [24]. Given a graph representing the information flow between nodes, the flow direction between certain nodes provides basic structural
information to the description of different motifs found in the application graphs. For example, consider the motif examples in figure 4.3. Each is size 3, meaning that 3 nodes are present in each. They are differentiated by the presence of and direction of traffic flow on the edges, represented by arrows.

4.1.3 Traditional Graph Measures

A graph measure is a metric which provides additional information about the graph to better describe its structure. Given the information graph, such measures have been used to classify network applications. Some graph measures describe the connectedness of a vertex, its distance from other vertices, its position in a graph, etc. The following sections describe various graph measures that are pertinent to this thesis along with examples.

Distances and Path Lengths

The distance between two nodes $u$ and $v$, written $d(u, v)$, is the length of the shortest path between them. In an unweighted graph, this is equal to the number of edges in the path. In a weighted graph, the length of path $P$ is $Pw(e)$ for $e \in P$. Dijkstra’s algorithm is one common method for determining this path through a network [16].

For a vertex $v$ in a connected graph, the eccentricity of $v$, $e(v)$, is the distance between $v$ and a vertex farthest from $v$ in $G$. The radius of a graph $\text{rad}(G) =$
\[
\min \{ e(v) | \forall v \in V \} \text{ and the diameter } \text{diam}(G) = \max \{ e(v) | \forall v \in V \}. \]

A vertex \( v \) is said to be central if \( e(v) = \text{rad}(G) \) and periphery if \( e(v) = \text{diam}(G) \) [16].

In Figure 3.1, the eccentricity of vertex 1, \( e(1) = 3 \) because the furthest vertex from 1 is vertex 6, and the distance between them is 3. The radius of the graph in figure 3.1 is 2 because \( e(3) = 2 \) and \( e(4) = 2 \), and these two vertices are tied for the minimum eccentricity in the graph. The diameter of the same graph is 3 as \( e(1) = 3 \), \( e(2) = 3 \), \( e(5) = 3 \), and \( e(6) = 3 \); each is tied for the maximal eccentricity in the graph.

**Centrality Measures**

Centrality measures in a graph indicate how relevant, important, or prominent a vertex is to the graph overall. The most basic centrality measure is degree centrality, or \( C_D(v) = \frac{\text{deg}(v)}{|V|-1} \). This equation can be modified for directed networks to produce \( C_Din \) and \( C_Dout \). In terms of social network analysis, indegree is interpreted as a measure of popularity, while outdegree is interpreted as gregariousness. In a dense adjacency matrix representation of a graph, the time required to calculate the degree centrality for all nodes is \( O(V^2) \), because all combinations of vertices must be considered [17] [10].

### 4.2 Graph Based Application Identification

Recall that graph-based application identification uses application graphs as tools to identify the network applications. The graphs usually model the network as host nodes labeled by IP address and edges between nodes representing some type of interaction between hosts. Graph-based identification involves identifying patterns of host behavior at the transport layer. This identification method is also deemed “in-the-dark” as it approaches the classification problem with two main constraints:
cannot use packet payload data and cannot rely on transport-layer port numbers for information about the application. Several such methods have been proposed including BLINC and Graption which are explained in detail in the following sections.

4.2.1 BLINC

BLINC is a graph-based classification approach short for BLINd Classification, named for its novel “in the dark” technique. It approaches the problem of classifying applications with the following constraints: i) no access to user payload is possible, (ii) well-known port numbers cannot be assumed to indicate the application reliably, and (iii) only information that current flow collectors provide can be used [21].

BLINC was the first method to shift the focus from classifying individual flows to associating specific IP address-labeled hosts with certain applications and then classifying the host’s applications accordingly. Observing the activity of a specific host provides more useful information with regard to discovering the behavior of the applications of that host. Moreover BLINC is novel in transforming the philosophy of identifying hosts to focus on three different levels, the social level, the network level, and the application level [21].

At the social level, the interactions of the host with other hosts is examined. At the network level, the behavior of the host is captured as the functional role of the host in the network. The functional role may be whether it acts as provider or consumer of a service. Finally, at the application level, BLINC captures the transport-layer interactions of the hosts on specific ports as a 4-tuple of (source IP address, destination IP address, source port, and destination port). The initial classification is then further enhanced by considering other metrics than transport-layer port number such as the transport protocol and the average packet size. Classifying the behavior of hosts on these three different levels provides increased knowledge of host behavior.
BLINC is novel in that it seeks to identify patterns of behavior of a host which aids in identifying which applications the host is most likely to be engaged in. Using these signature patterns, BLINC has been successfully applied to several real datasets with accuracy between 80% - 90% [21]. Furthermore, BLINC has been shown to be capable of detecting unknown applications as well as potentially malicious flows which deviate from the signature behavior. It is important to note that these last two features are impossible with traditional transport-layer port number-based identification.

### 4.2.2 Graption

Graption (Grap h-based classification) is another graph-based classification framework originally proposed in 2008 as a systematic way to combine network-wide behavior with flow-level characteristics of network applications. It models this behavior
through graphs with IP address hosts as nodes and edges representing some interaction between hosts, referring to these graphs as Traffic Dispersion Graphs (TDGs). The idea is that TDGs enable the detection of network-wide behavior common among P2P applications but different from other traffic such as Web [19].

Utilizing flow-level features, Graption first groups flows in an unsupervised way, without using application-specific knowledge. It then uses TDGs to classify each group of flows. Graption uses real-world backbone traces and derives graph theoretic metrics enabling the distinguishing of P2P TDGs and client-server TDGs such as those of Web traffic. The difference between P2P TDGs and client-server TDGs can be visualized by comparing figures 3.5 and 3.6. Graption is also a very practical system, noting that a single backbone link is sufficient to generate TDGs that can be
used to classify traffic and that TDGs of the same system are fairly consistent over time. It is also highly accurate for P2P classification capable of classifying 90% of P2P traffic with 95% accuracy. Compared to BLINC [21], Graption is much more accurate with regard to P2P traffic. For example, Graption correctly identifies over 95% of BitTorrent traffic compared to BLINC’s mere 25% [19].

Whereas BLINC and Graption both attempt to classify applications based on overall graph structure, this thesis shows that analyzing motifs, the basic substructure building blocks of graphs, can lead to even better results.
4.3 Motif Approach to Application Classification

Another identification method proposed by my peers at Wake Forest was developed by borrowing ideas from biological networks. Cellular processes are regulated by the complex interactions of several molecules such as proteins and DNA [32]. After modeling these interactions as graphs, this particular approach then searches the graphs for *motifs*: recurring, significant patterns of interconnections. Motifs occur in biochemistry, neurobiology, ecology, and computer networks, among others [24] making motifs the most basic structural elements capable of defining broad classes of networks.

Allan’s approach was the first to analyze motif presence in application graphs as a useful method of application classification. He determined that it is important to use the right sized motifs in the analysis as motifs that are too small (ie. 1 or 2 nodes) add little to no information gain, and motifs that are too large are too computationally expensive to search for. Motifs that are too large may also lead to over-fitting.

Pioneering the motif-based classification approach, Allan was able to achieve better results than a traditional graph measures approach. In the end, his single-port analysis was able to achieve classification accuracy of 85.7% compared to 79.54% of the traditional graph measures [16].
In today’s environment of rapidly evolving network applications, it is unrealistic to assume that an application only uses one port. Many applications use more than one port. In fact, many of the applications studied by Allan use multiple ports, but Allan only analyzed information that travelled on a single port. Considering multiple ports will provide more data to analyze and therefore increase classification accuracy.

5.1 Applications and the Use of Multiple Ports

The reasons certain applications use multiple ports are quite diverse. Many applications separate various facets or features of their operations into separate ports. For example, it is common for an application to allocate file-transfer traffic, audio traffic, and video traffic to different ports. This is often done so that disrupting one connection does not impact others and the application can still operate. Some applications use one port as a control connection and a different port for data transfer [14]. Others not only use multiple ports but also multiple application-layer protocols. For example, the P2P file-sharing application KaZaA uses its own proprietary protocol for signaling and the HTTP protocol for its file-transfer service completion [30].

It is also common for many applications to generate HTTP and DNS traffic on ports 80 and 53 respectively in addition to a domain specific port chosen by the application. This traffic may involve resolving domain name lookups, file downloads, advertising, etc. For example, MSN Messenger provides a communication service through its proprietary protocol for instant messaging, voice chatting, etc. on large-numbered and dynamically allocated ports, but it also simultaneously generates HTTP traffic.
via the ad banner embedded in its chatting windows which is not directly related to any communication use [30].

The aim of this thesis is to extend functionality provided by the foundation of previous motif-based identification research by Allan to classifying high-level applications instead of single-port protocols. There is also potential for this research to be more practical for real world application-identification as the ability to label traffic as high-level applications (such as Web, Dropbox, Dota 2, etc.), is much more powerful than the ability to label low-level protocols such as HTTP, DNS, or FTP. Many different types of application use these protocols, and the network administrator is more interested in discovering how they are being used at a higher level rather than the simple fact that these standard protocols are currently running on the network.

A classic example of a multi-port application is FTP. In this protocol, when a client wants to connect to either upload or download files, it makes a TCP connection on port 21. This connection carries all of the interactive user traffic as well as control commands such as changing directory. Additionally, it uses this control connection to tell the server which port it wishes to use for transferring data, typically a high-numbered TCP port [14].

When a user wishes to transfer a file, he sends a PUT or GET command to the server on port 21. The server then makes a new TCP connection to the previously mentioned high-numbered port on the client agreed upon earlier through the control session. The source port for this connection is port 20, the FTP data port. Interestingly enough, this is the opposite sequence of most TCP connections by which the client device usually connects to the server using a well-known destination port number. Here the server is actually connecting to the client through a well-known source port number. The client and server proceed to exchange the file and subsequently disconnect this FTP data connection, leaving the control connection on port 31.
5.2 Using Motifs for Application Identification

The overarching idea of this thesis is that highly significant motifs serve a specific function in computer networks that makes them an ideal indicator of that application’s behavior. Figure 5.1 describes the overall process from data collection to application classification. Essentially, the process involves generating and capturing ground-truth data, storing the data in the database, constructing application graphs, searching for the relevant motifs for each application, and then classifying the applications based on the vertex profiles for each application. The next few subsections will describe each step in detail.

5.2.1 Data Collection

This research used Cisco NetFlow flow data for modeling host-to-host communication. Recall that the data fields provided by Netflow are summarized in chapter 2. The most relevant to this thesis are source IP address, destination IP address, source port, destination port, number of bytes sent, and duration of the connection.

This data provides summary information collected from packet headers only. A connection is uniquely identified by the source IP address, destination IP address, source port, destination port, and start time. This NetFlow data is automatically collected by all Cisco networking devices, meaning that no additional steps are necessary other than simply retrieving the records from the Cisco devices. This is a major
advantage of this method and “in-the-dark” methods in general as opposed to deep packet-inspection methods. Network managers usually do not collect Pcap data, detailed logs of every packet that crosses the network because it is infeasible due to the size requirements. However, since flow data only maintains a few packet summary fields it is much more feasible to log.

5.2.2 Application Graphs

Figure 5.2: At left is an example application graph that consists of 15 nodes, while at right is a motif of size 3 that occurs 5 times in the application graph. Node 6, involved in the size 3 motif with nodes 5 and 10, also interacts separately with node 9. [9]

An application graph is a high-level modeling of an application’s host-to-host communication. Each node in the application graph represents a distinct IP address. The directed edges are used to show the flow of data between two hosts. Figure 5.2 shows an application graph and a size 3 motif that occurs 5 times in the graph. Considering the small size of the application graph in this example, this motif is very significant to identifying the overall graph’s structure. This is the basic premise of the thesis. Several applications use the same motifs but overall have easily differentiated
application graphs.

**Combining all Ports into One Graph**

Before creating a process which uses multiple ports in application identification, the relevant ports for each application must be discovered. All of the applications studied in this thesis use many more ports than are feasible to consider in the classification. For example, the traffic trace for World of Warcraft uses over 1600 ports. All the applications use at least a couple hundred ports. Using that many ports in classification would make the process intolerably slow since an application graph must be constructed for each individual port, and it would risk overtraining to a very specific system that is running that application.

Instead of using every port that an application employs in the classification process, only ports 80 and 53 are combined as they are universally used in every application studied and provide sufficient information together to achieve very high classification accuracy.

In order to use multiple ports in the classification process, many different graphs of individual ports must be combined into one combined-port-graph of all the ports relevant to classification for that application. Once the ports have been combined, then the motif analysis can be performed.

### 5.2.3 Vertex Profiles

After constructing the combined-port graphs and discovering the relevant motifs therein, vertex profiles are created to succinctly model the application’s multi-port behavior according to various metrics. A vertex profile is defined as a datapoint in a $d$-dimensional space, where $d$ is the number of attributes $a$ in the profile. The attributes $a_1$ through $a_d$ can be any numerical representation of a datatype [16]. The
idea is to associate an application with a specific profile. Figure 4.X describes a list of \( n \) vertices in a vertex profile.

\[
v_1 = [a_1, a_2, a_3, \ldots, a_d]
\]

\[
v_2 = [a_1, a_2, a_3, \ldots, a_d]
\]

\[
\vdots
\]

\[
v_n = [a_1, a_2, a_3, \ldots, a_d]
\]

Figure 5.3: A List of \( n \) Vertices in a Vertex Profile [16]

In the motif-based approach, \( d \) is equal to the total number of motifs discovered in the combined-port graph. A binary attribute is created to designate whether or not the vertex participates in the motif. As part of the toolchain, FANMOD outputs a comma separated value file of the format:

```
adjacency matrix, [participating vertices]
```

This file is parsed and the data is used to create the profiles for each node based on its participation in significant motifs. In this case, \( d \) is \( X \) with \( Y \) of these coming from order 3 motifs, and \( Z \) of these coming from order 4 motifs. Motifs smaller than order 3 are too small to capture any real information about application behavior, and motifs larger than order 4 are too large to efficiently find as the motif search algorithm is exponential by the size of the motif input variable. Preliminary results suggest that using both order 3 and order 4 motifs together produces better results than either one does individually [16].

To further improve the results, the nodes can be colored according to specifications of other features relevant to an application’s behavior. These other features are described below:

1. **Node Type:** colors each host as client, server or peer. A node is a client if it
only sends data, a server if it only receives data, and a peer if it both sends and receives.

2. **Node Connections:** colors each host based on the number of connections it has to other hosts. Colors are split into 6 approximately equally sized bins.

3. **Node Duration:** colors each host based on the average length of time a connection is live. Colors are split into 6 approximately equally sized bins.

4. **Node Bytes:** colors each host based on the total number of bytes sent. Colors are split into 6 approximately equally sized bins.

Using these additional features adds a significant amount of information about the application’s behavior that proves to be highly relevant to classifying the application.

### 5.3 Implementation of Motif-Based Application Identification

The previous section provided the theoretical approach to classifying applications. This section will provide the implementation details associated with each step in Figure 5.1.

After obtaining the datasets, the first step in the toolchain is to parse the packet capture (pcap) files. Once parsed, the packets are stored in a MySQL database. The next step is to create the application graphs and analyze them according to the traditional graph measures described previously [16].

Next, the application graphs were searched for motifs with FANMOD (**Fast Network Motif Detection**). First, network data is retrieved from the database and formatted as FANMOD input. Then FANMOD searches for motifs of size 3 or 4. Next, randomized graphs are generated, and FANMOD then searches these randomized graphs
for motifs of size 3 or 4. After comparing motifs from the randomized graphs to the motifs found in the application graphs, those that appear in the application graphs significantly more often than in the randomized graphs are returned to the user [16].

Next, vertex profiles are generated from the data collected through the eleven node characteristics and the motif analysis described above. These profiles combined the order 3 and order 4 motifs together.

Figure 5.4: First section of the toolchain

The implementation of the process used is a modified version of previous work completed by Eddie Allan and Chaz Lever. Before the toolchain can be run, there are a few prerequisites including obtaining datasets and converting datasets to sql statements that can be inserted into the MySQL database. The applications used in this thesis were all captured using tcpdump. These captures were then converted to flow data and inserted into the MySQL Motifs database using a python script, pcap2silk2mysql.py, originally written by Chaz Lever.

Once each of the example application traces has been captured and stored in the database, the toolchain can be started to process the data and retrieve the classification results. Figure 4.4 illustrates the first step to the toolchain, converting the data stored in the Motifs database to FANMOD input. This section executes mysql2fanmod.py on each application to be considered for classification. The -f pa-
rameter designates which feature to use in the classification. Here it is set to nodetype, which only considers the presence or absence of each of the motifs discovered during the search phase. The -u parameter designates how many rows to pull from the database table. This option was more useful in previous research which used outside datasets which were very large and thus slow to classify unless only a subset of the data was extracted from the table.

The -o parameter specifies where to place the output of this script which is actually input to FANMOD. The -p parameters specify which additional ports to consider in addition to the standard port. The standard port is the very last command line argument on each line. In this case, it is always port 80 as that port was universally used in the studied application set. Between the -p options and the standard port argument is the database table in the Motif database to be classified. By convention, the table name is always the lowercase name of the application.

## run fanmod on the data
```
..toolchain/run_fanmod.py --size=3 /home/bailey/testRun/miniyahooch_443_53_80.IN
..toolchain/run_fanmod.py --size=4 /home/bailey/testRun/miniyahooch_443_53_80.IN
..toolchain/run_fanmod.py --size=3 /home/bailey/testRun/diablo2_443_53_137_138_4000_6112_6113_80.IN
..toolchain/run_fanmod.py --size=4 /home/bailey/testRun/diablo2_443_53_137_138_4000_6112_6113_80.IN
..toolchain/run_fanmod.py --size=3 /home/bailey/testRun/dota2_443_53_137_138_35_44_80.IN
..toolchain/run_fanmod.py --size=4 /home/bailey/testRun/dota2_443_53_137_138_35_44_80.IN
..toolchain/run_fanmod.py --size=3 /home/bailey/testRun/counterstrike_443_53_137_138_9980_80.IN
..toolchain/run_fanmod.py --size=4 /home/bailey/testRun/counterstrike_443_53_137_138_9980_80.IN
..toolchain/run_fanmod.py --size=3 /home/bailey/testRun/wow_53_137_138_3724_688_1_80.IN
..toolchain/run_fanmod.py --size=4 /home/bailey/testRun/wow_53_137_138_3724_688_1_80.IN
..toolchain/run_fanmod.py --size=3 /home/bailey/testRun/miniskype_443_53_137_138_67_68_2040_80.IN
..toolchain/run_fanmod.py --size=4 /home/bailey/testRun/miniskype_443_53_137_138_67_68_2040_80.IN
```

Figure 5.5: Second section of the toolchain
Figure 4.5 shows the second step in the toolchain. This step involves running FANMOD on the output of the last script for each application. This script is very straightforward. It searches each input file, the .IN files, for both size 3 and size 4 motifs. Remember that previous work has established that using motifs of size 3 and 4 together generates better results than either does alone [16]. This part of the toolchain is the most time consuming. FANMOD is an expensive process, and its complexity is exponential to the size of the motifs it searches for. This ultimately means that it is impractical to consider motifs larger than size 4.

```
## get important motifs
../toolchain/motif_results.py -o /home/bailey/testRun -r /home/bailey/testRun
## build motif profile
../toolchain/motif_profiles.py -r /home/bailey/testRun /home/bailey/testRun > /home/bailey/testRun/motif_analysis.dat
ls -l /home/bailey/testRun/*/mappings > mappingFile.txt
java MergeSeparatePorts mappingFile.txt /home/bailey/testRun/motif_analysis.dat
```

Figure 5.6: Third section of the toolchain

The next step in the process is shown in Figure 4.6. This step involves analyzing the motifs to determine the most important ones in motif_results.py, and then building the motifs with motif_profiles.py. Both of these scripts were also written by Chaz Lever. Motif_profiles.py produces a file called motif_analysis.dat, which
Chapter 6: Experimental Results

This chapter will compare and contrast the new multi-port classification approach proposed in this thesis to the past approaches of single-port motif-only classification and single-port multi-feature classification.

6.1 Network Traces

Each classification method will be tested using network traffic collected from seven applications: Diablo 2, Dota 2, World of Warcraft, Counterstrike, Skype, email, and web browsing. A very common issue with traffic classification research is that labeled (ground-truth) anonymized traffic traces are largely unavailable. This is further complicated when multiple ports need to be associated with a single application. For example, consider a user browsing the web while using an email client. Since both applications use Domain Name Service (DNS, which is a separate port), it is difficult to determine whether the application may have used DNS at any particular point in time. In order to construct a fair comparison, each method must be run over the same datasets.

The network traces used in this thesis are all generated in-house. The structure of this research requires ground-truth datasets because there is no clear method of separating individual multi-port applications’ traffic in a large dataset containing many applications running concurrently. Previous research used these outside network traces because they operated under the assumption that they could label all traffic under a particular port as a single protocol. For example, they labeled all traffic through port 80 as simply HTTP protocol and likewise all traffic through port 53 as DNS. It is true that traffic through these ports uses these protocols, but all of the
higher-level applications that are classified in this thesis use both HTTP and DNS as part of their operation, and the previous assumption that a single port can be used as a ground truth indicator of a particular application is no longer valid. Therefore traffic cannot simply be pulled from these large outside datasets as it has been in the past, and instead it must be generated in-house.

6.1.1 Sources of Traces

The following gives a brief overview of each application chosen for research. These network applications were selected to be representative of the types that could be found in many networks (web, games, voice over IP, and email).

Email Client

The Mozilla Thunderbird email client was chosen for study. Thunderbird is a free, open source, cross-platform email client written primarily in C++ [4]. A script was written to read and write emails between dummy accounts. The bot included features such as reading for random amounts of time and deciding to read or write via a coin flip for that session in an attempt to better model actual human user behavior. Since the primary goal is to determine the traffic as a generic label of email client rather than the specific client, the data from Thunderbird was mixed upon being inserted into the database.

Web

Web traffic was generated in the Google Chrome browser. A few different bots were written to simulate user behavior: a Gchat bot that sends and receives short chat messages between two dummy accounts, a Gmail bot that sends and reads emails between two dummy accounts, and a YouTube bot which randomly browses YouTube
videos for random amounts of time. Although web traffic is immensely varied impossible to perfectly summarize for a research study of this magnitude, these three different uses represent a variety of web use that is at least somewhat representative of a typical internet user. The data from these different sources was also mixed upon being inserted into the database.

**Skype**

Skype is a proprietary Voice-Over-Internet Protocol (VoIP) service and software application owned by Microsoft Corporation. It allows users to communicate with peers via voice, video, and instant messaging over the internet. It is a hybrid peer-to-peer and client-server system. Traffic was also collected by an AutoIt bot simulating user behavior.

**Diablo II**

Diablo II is an action role-playing video game developed by Blizzard Entertainment and released in the year 2000. It runs on both the Microsoft Windows and Mac OS operating systems, but traffic was only captured of the game running on Windows. An AutoIt bot was written to play the game in a way that should be fairly representative of an actual gamer. The script opened up games, waited a random amount of time, then left the game and started a new game. Diablo II is the oldest game used in this research.

**World of Warcraft**

World of Warcraft is a Massively Multiplayer Online Role-Playing Game (MMORPG) also developed by Blizzard Entertainment and released in the year 2004. Traffic was captured by a bot playing the game on the Windows operating system. It is the
most widely played MMORPG in the world, with over 10 million subscribers as of February, 2012 [8]. The bot logged into the game server and walked around randomly observing other players.

Counter-Strike

Counter-Strike is a tactical first-person-shooter (FPS) video game developed by Valve Corporation and released in 2003. It runs on both Microsoft Windows and the Microsoft Xbox game console, though traffic for this research was only captured of the game running on Windows. An AutoIt bot was also written to simulate user behavior. The bot walked around randomly in the map and fired bullets sporadically. It did not compete very well using this strategy, but it may emulate an unskilled player fairly accurately.

Dota 2

Dota 2 is an upcoming action real-time strategy (RTS) video game currently in Beta being developed by Valve Corporation. A beta key, an early-access mechanism to a yet-to-be-released video game, was obtained by co-researcher Brad McDanel, and an AutoIt script was written to simulate a gamer playing the game. The bot joined a game and observed other players, both teammates and opponents battling.

It is important to note the high variety chosen among different types of applications. Within the gaming category, both very old games and cutting-edge not yet released games were studied as well as vastly different types of games. This variation is indicative that the approach proposed in this thesis can be universally applied to all different types of network applications.

Once the bots were written, capturing the network traffic was completed by utilizing tcpdump. Each iteration captured one hour of traffic for one particular application, separated into four 15 minute segments. The data captured was in Pcap format,
ready to be inserted into the database with the python script, pcap2silk2mysql.py, described in the previous chapter.

6.2 Experimental Set-up

As shown in figure 5.7, the classification process involves network traffic generation, data parsing and storing into the database, creation of graphs and vertex profiles, node property analysis, motif searching, combination of individual port graphs into a mega-graph, combination of multiple node profiles, and SVM analysis. Once the data has been processed, the application graphs are created by querying the MySQL database table for each specific application for all entries in which either the source or destination port matches one of the multiple ports previously determined to be relevant to the identification of that application. Since ground-truth datasets were used, it is known that the data pulled from the table in this way is actually representative of the network application in question.

Although each dataset represents one hour of network traffic for each particular multi-port application, the sizes of the corresponding application graphs vary widely based on the nature of the application. Media applications are much more data intense compared to some games which only submit short status text messages between server and client. Two tables (web and Skype) actually had to be reduced to miniature size in order for the toolchain to complete processing in a reasonable amount of time. Some applications make many more connections than others:

- Web: 1000 connections
- Diablo 2: 3836 connections
- Dota 2: 2016 connections
- Counterstrike: 709 connections
- World of Warcraft: 3241 connections
- Skype: 2000 connections

The number of connections alone is not the only determining factor of computational complexity, but it is a reasonable indicator. In the end, ports 80 and 53 were chosen for analysis as every application studied used at least these two ports. Moreover, the results obtained from combining these two ports show that they are an excellent combination for highly accurate classification.

### 6.2.1 Metrics Used to Measure Performance

Classification results are displayed as confusion matrices. Each column in a confusion matrix represents the instances in a predicted class, while each row represents the instances in an actual class. The bolded numbers along the diagonal indicate the correct classifications. Confusion matrices show two different types of misclassifications, false positives and false negatives. False positives are data points found in the row of a specific class that are predicted to have the class label but are incorrect. On the other hand, false negatives are examples of a particular class that are incorrectly labeled, shown in the column of a class.

![Table 6.1: An example confusion matrix with 3 classes](image)

Table 5.1 displays an example confusion matrix with three classes. In this example, of the 8 actual members of class A, the system predicted that 3 were class B, and of
the 6 actual members of class B, it predicted that 2 were class A and 1 was class C. This system seems to have trouble distinguishing between class A and class B but is much more accurate at correctly predicting class C. All the correct predictions are displayed in bolded font along the diagonal of the table.

The primary metric to measure classification performance is accuracy. The overall accuracy of a model is the number of correct classifications (true positives) over all classifications. For each class, accuracy is the number of true positives for that class over the total number of examples of that class.

To test for generalization error, 10-fold cross validation is performed. In 10-fold cross validation, the training data is randomly split into 10 mutually exclusive subsets of approximately equal size. The SVM is then trained on 9 of the subsets and tested on the remaining subset. This process is repeated so that each of the 10 subsets has a chance to be the test subset. Averaging the test error over 10 trials gives the generalization error. 10-fold cross validation is often considered a good approach because it allows training and testing on the entire dataset, meaning that the dataset doesn’t have to be split explicitly into a training set and testing set which effectively halves the available data [15].

6.3 Results and Analysis

In order to discover the usefulness of the new approach proposed in this thesis, experiments were run on the toolchain described in detail in chapter 4 and then compared to results generated from a modified toolchain that emulates Allan’s and Lever’s work. Each approach relies on motif analysis of application graphs and uses an SVM for classification. Recall the following major differences between each of the three approaches:

1. Allan’s Approach: single-port analysis, single feature (node type)
2. Lever’s Approach: single-port analysis, multiple features (node type, node bytes, node duration, node connections)

3. Bailey’s Approach: multi-port analysis, multiple features (node type, node bytes, node duration, node connections)

The approaches vary on the other features used besides motif structures found in the graph. These features are here defined:

- **Node Type**: colors each host as client, server or peer. A node is a client if it only sends data, a server if it only receives data, and a peer if it both sends and receives.

- **Node Connections**: colors each host based on the number of connections it has to other hosts. Colors are split into 6 approximately equally sized bins.

- **Node Duration**: colors each host based on the average length of time a connection is live. Colors are split into 6 approximately equally sized bins.

- **Node Bytes**: colors each host based on the total number of bytes sent. Colors are split into 6 approximately equally sized bins.

Below are the results of the three different approaches:

<table>
<thead>
<tr>
<th></th>
<th>C-S</th>
<th>Diablo II</th>
<th>Dota 2</th>
<th>Skype</th>
<th>Email Client</th>
<th>Web</th>
<th>WoW</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Diablo II</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Dota 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>86.1%</td>
</tr>
<tr>
<td>Skype</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Email Client</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>0%</td>
</tr>
<tr>
<td>Web</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>WoW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Total Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63.9%</td>
</tr>
</tbody>
</table>

Table 6.2: Single-port analysis, only node type feature results
The single port with node type feature only considered port 80 for each application. This is not a perfect representation of Allan’s approach because in reality he used a different single port for each application. He obtained this port through the well-known port list. Since some of these applications are not maintained on that list, port 80 was used for every application. Additionally, the original method for the single port node type feature used k Nearest Neighbor (kNN) for classification, while the results presented in this thesis use an SVM.

<table>
<thead>
<tr>
<th></th>
<th>C-S</th>
<th>Diablo II</th>
<th>Dota 2</th>
<th>Skype</th>
<th>Email Client</th>
<th>Web</th>
<th>WoW</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-S</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Diablo II</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>27.3%</td>
</tr>
<tr>
<td>Dota 2</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>92.3%</td>
</tr>
<tr>
<td>Skype</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>36</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Email Client</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Web</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>WoW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Total Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>94.6%</td>
</tr>
</tbody>
</table>

Table 6.3: Single-port analysis, multiple features results

The single port, multiple feature approach approach was also simulated by using port 80 for each application. Originally, this approach also used a differing single port for each application, but this could not be perfectly simulated for the same reasons as the single port with node type feature approach. This method is very accurate for everything except for Diablo II. It does not simply misclassify Diablo II as a single different application as the errors are approximately equally spread across Dota 2, Skype, and WoW. This is very different from the single port with node type feature results. Diablo II is the oldest game studied and was probably built under a simpler client-server model than the other games which confuses the classifier. The next result set clearly shows the benefit of using multiple ports in analysis for Diablo II.
Table 6.4: Multi-port analysis, multiple features results

<table>
<thead>
<tr>
<th></th>
<th>C-S</th>
<th>Diablo II</th>
<th>Dota 2</th>
<th>Skype</th>
<th>Email Client</th>
<th>Web</th>
<th>WoW</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-S</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Diablo II</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Dota 2</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>92.9%</td>
</tr>
<tr>
<td>Skype</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Email Client</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Web</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>WoW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>100%</td>
</tr>
<tr>
<td>Total Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.4%</td>
</tr>
</tbody>
</table>

The Multi-port results were obtained by using ports 80 and 53 combined for every application. The number of examples is therefore slightly larger than each of Allan’s and Lever’s because port 53 adds a few examples to significantly improve performance.

Tables 5.2, 5.3, and 5.4 show the results for each of the approaches previously discussed. The largest jump in overall accuracy, from 63.9% to 94.6%, occurred when adding the multiple features into consideration instead of solely focusing on node type (client, server, or peer coloring). These multiple features provide much more information about the graph’s structure. For example, the node bytes feature is able to distinguish similarly connected motifs by how much data is actually being transferred between the nodes. This greatly assists with distinguishing data-intensive applications from others. Moreover, node duration can similarly assist with distinguishing applications that sporadically connect and disconnect with those that leave connections open for longer periods of time. For example, the web browsing script constantly visits new webpages, opening and closing connections relatively quickly, whereas most of the game scripts open long connections to the game server which only terminate upon the end of play.

The approach discussed in this thesis of multi-port, multi-feature analysis tweaked Lever’s previous version to near perfection at 99.4%, misclassifying only one example of Dota 2 as Skype. This clearly shows the benefit of using multiple ports in the anal-
ysis process. Distinguishing between just these 7 applications was nearly perfect with utilizing the information from just two ports, but more ports may be necessary for scalability to a larger application set. In theory it would seem to be more difficult to classify with a one versus all SVM if the number of applications were in the hundreds or thousands instead of 7. Therefore, there would probably be a greater increase in multi-port analysis performance over single-port analysis performance as the number of applications in the classification set grew in number. Additionally, more ports would most likely be needed to achieve the same level of classification accuracy seen in these results.
Chapter 7: Conclusions and Future Work

As network communications and applications become more complex, operating, managing and securing networks have become increasingly challenging tasks. Identifying the applications on a network has become very difficult due to the use of non-standard port numbers and packet encryption. Thus, a new method is needed for application discovery. This thesis has presented a new approach to this problem using multi-port analysis. It extends the great work done by Wake Forest colleagues Eddie Allan and Chaz Lever to achieve better results by combining the information from multiple ports of an application into a central application graph for motif and statistical analysis.

7.1 Network Application Identification

Identifying the specific applications running on a computer network is of utmost importance for quality of service and other resource management, network planning, network security, and usage policy enforcement. For example, determining the resource-intensive applications currently running such as video-conferencing (Skype) is important in order to provide sufficient bandwidth to the user to support such applications. Additionally, knowing the full range of applications using the network can assist with detecting security breaches such as botnet activities and other intrusions. This knowledge can also enforce usage policy by effectively banning applications commonly used for software and other multimedia piracy. There are a countless number of examples of the need to fully understand the types of applications that can provide network administrators detailed knowledge of the all of the specific applications currently utilizing the network’s resources.
In chapter 2, the pros and cons of various methods to identify network applications were discussed. Transport layer port-based identification is the simplest of these methods, but is quickly becoming obsolete as applications no longer use well-known port numbers or tunnel through a port, using it differently than the specification intended. Signature-based methods classify based on deep packet inspection. This method is easily evaded by packet encryption. Statistics based methods analyze details such as packet size, packet inter-arrival times, and connection durations. This method has been shown to be fairly accurate by Andrew Karode [28], but also requires access to such low-level details. Finally, graph-based methods such as Blinc Blinc and Graption [19] model an application’s behavior in a high-level application graph and classify based on traditional graph measures such as distance, centrality, and clustering coefficient. The approach taken in this thesis combines the best aspects of graph-based approaches and statistics-based approaches and does not require packet-level information.

7.2 Motif-Based Application Identification using Multiple Ports

The approach developed in this thesis to identify network applications is an extension and enhancement of previous motif research of Allan and Lever. After selecting the applications for study, the most relevant ports representing each application’s behavior are selected by the previously discussed metrics (highest number of connections, most data sent, and highest duration). First an application graph is constructed for each relevant port to each application. These graphs are then mined to discover the significant motifs, patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks [24]. Next these individual port graphs are combined to create a mega-graph for each application of
all the ports that are considered to be relevant to defining its overarching behavior. Finally the motif data is sent to the Support Vector Machine (SVM) for classification.

Table 6.4 summarizes the results of the approach taken in this thesis. The results are very promising, with an overall accuracy of 99.4%, and represent a significant increase in performance over both Allan’s and Lever’s previous work.

7.3 Areas for Future Work

The topic of application discovery is quite large, and there exists a plethora of venues for future work to further the development of the approach described in this thesis. One such venue is testing the method on more application types. Another area of interest for practical concerns is to augment the method so that applications can be identified in real time. Currently, the method only works for offline ground truth datasets that are captured as the sole application running. Taking the theoretical approach in this method and applying it to a large traffic mix with many applications would be a difficult but very rewarding area of future research.

7.3.1 Using Other Ports

This thesis ultimately combined information from only ports 80 and 53 for classification. Studying only 7 different applications, the results proved to be very good, but it is yet to be determined whether the high accuracy is maintainable as the number of applications to be classified scales by orders of magnitude. Since the classification approach in this thesis uses multiple passes of a one versus all SVM, it will be much harder to distinguish individual applications if the number of applications were in the thousands, for example. In such a larger scale approach, it may be necessary to use information from other ports in addition to 80 and 53.
7.3.2 Tor and Onion Routing Issues

It is important to note problems occurring as a result of onion routing. *Onion routing* is a technique for anonymous communication over a computer network in which messages are repeatedly encrypted and then sent through several network nodes called onion routers. Each onion router removes one layer of encryption to uncover its routing instructions and sends the message to the next router where the process is repeated until the message reaches its final destination. The purpose is to prevent these intermediary nodes from knowing the origin, destination, and contents of the message.

The problem is that Tor and Onion aggregate and encapsulate multiple connections into one connection via tunneling. As a result, the approach presented in this thesis would only see one connection. A Tor or Onion router would at some point break the single communication into multiple communications once it was closer to the destination. This limits the traffic collection capability to one hop.
Bibliography


modeling of computer systems, SIGMETRICS ’09, pages 49–60, New York, NY, USA, 2009. ACM.


Curriculum Vitae

• Education
  – Wake Forest University 2010 - 2012
    * M.S. in Computer Science
    * Specialization in Computer Network Security
  – Wake Forest University 2006 - 2010
    * B.S. in Computer Science
    * Concentration in Mathematics, Economics
    * Summa Cum Laude, Phi Beta Kappa

• Employment History
  – Wake Forest University (August 2010 - May 2012)
    * Graduate Teaching Assistant and Research Assistant
    * Part of the motif research group seeking to secure computer networks
    * Contributed to the development of a multi-port network application identification method
  – Ednovo (May 2011 - August 2011)
    * Software Engineering intern at Ednovo, a non-profit ed-tech startup
    * Helped develop the frontend and backend systems for Gooru
    * Gooru is a free education platform aiming to provide first-rate education to the world’s 1 billion students within 3 years
  – EvaBank (January 2010 - August 2010)
* Assistant account executive and IT support

* EvaBank is a small bank in my hometown of Birmingham, Alabama