

3-11-2011

A Multi Agent System for Flow-Based Intrusion Detection Using Reputation and Evolutionary Computation

David Hancock

Follow this and additional works at: <https://scholar.afit.edu/etd>

Part of the [Computer and Systems Architecture Commons](#), [Digital Communications and Networking Commons](#), and the [Information Security Commons](#)

Recommended Citation

Hancock, David, "A Multi Agent System for Flow-Based Intrusion Detection Using Reputation and Evolutionary Computation" (2011). *Theses and Dissertations*. 1391.
<https://scholar.afit.edu/etd/1391>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.



**A MULTI AGENT SYSTEM FOR
FLOW-BASED INTRUSION DETECTION
USING REPUTATION AND
EVOLUTIONARY COMPUTATION**

THESIS

David L. Hancock, Captain, USAF
AFIT/GCS/ENG/11-02

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT/GCS/ENG/11-02

**A MULTI AGENT SYSTEM FOR FLOW-BASED
INTRUSION DETECTION USING REPUTATION
AND EVOLUTIONARY COMPUTATION**

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science

David L. Hancock, BCEN
Captain, USAF


March 2011

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**A MULTI AGENT SYSTEM FOR FLOW-BASED
INTRUSION DETECTION USING REPUTATION
AND EVOLUTIONARY COMPUTATION**


David L. Hancock, BCEN
Captain, USAF

Approved:



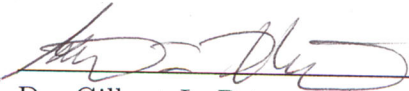
Dr. Gary B. Lamont (Chairman)

16 MARCH 2011
Date



Dr. Barry E. Mullins (Member)

16 Mar 11
Date



Dr. Gilbert L. Peterson (Member)

16 MAR 2011
Date

Abstract

Intrusion Detection (ID) is essential for protecting computer networks. The rising sophistication of threats as well as the improvement of physical network properties present increasing challenges to contemporary ID techniques. For example, the rate of traffic inflow prevents most traditional network- and signature-based ID systems from conducting more than a sparse sampling, opening the door for malicious traffic to enter the network without scrutiny. ID techniques are commonly defined in terms of *location* (i.e. *where* it is performed) as well as *approach* (*how* it is performed). With respect to location, the *multi agent* design paradigm leverages the strengths of both *network*-based and *host*-based ID methods. With respect to the approach, *flow*-based intrusion detection complements traditional *signature*-based and *behavior*-based ID systems.

This research develops: 1) a scalable software architecture for a new, self-organized, multi agent, flow-based intrusion detection system; and 2) a network simulation environment suitable for evaluating an implementation of this multi agent system (MAS) architecture and for other network research purposes.

Self-organization is achieved in two ways. First, a “reputation” system permits agents to dynamically find nodes that are most effective for classifying malicious network activity. Second, multi objective evolutionary algorithms aid in the search for effective operational parameter values.

A first implementation of the MAS architecture using reputation is quantitatively evaluated, via hypothesis testing, and found to significantly outperform a static, randomly distributed MAS for certain combinations of agent population sizes and observation periods. Improvements range from 3.6 to 11.6 percent increased classification

accuracy.

Following these results, the network simulation environment complexity is increased in a second iteration design, and a new MAS is developed to deal with more realistic challenges, including communication difficulties and a broader range of malicious activity scenarios. The environment and the MAS are qualitatively evaluated and found to successfully achieve essential functionality.

These encouraging results establish an optimistic outlook for further research in flow-based multi agent systems for intrusion detection in complex computer communication networks.

Acknowledgements

I express my deep gratitude to those on Earth as well as Those above that supported my efforts to develop this thesis.

My spouse and my children patiently and gracefully handled my time away from home throughout this assignment. I owe them my heartfelt thanks for providing smiling faces at the end of many difficult days. They helped place this effort in its proper context, and I look forward to happy days ahead facing struggles together as we did at AFIT.

Professor Lamont also demonstrated patience, encouragement, and self-sacrifice in advising me throughout. He has been one of my strongest advocates in six years of service in the Air Force, and I am indebted to him for his efforts. I also thank Dr. Gilbert Peterson and Dr. Barry Mullins for valuable feedback on the final document and outstanding academic instruction which inspired it.

I thank my classmates for being Wingmen in the true sense of the word.

God lives and loves us. Of that I bear personal witness.

David L. Hancock

Table of Contents

	Page
Abstract	iv
Acknowledgements	vi
List of Figures	x
List of Tables	xii
List of Abbreviations	xiii
I. Introduction	1
1.1 The Generic Intrusion Detection Problem	2
1.2 Scope of Investigative Domain	4
1.3 Research Goal and Evaluation Hypothesis	6
1.4 Investigative Approach	7
1.4.1 Prospective Beneficiaries	8
1.4.2 The Use of Reputation	9
1.5 Thesis Overview	10
II. Literature Review	12
2.1 Internet Modeling	13
2.1.1 Modeling Limitations	14
2.1.2 Modeling Internet Topology	15
2.1.3 Modeling Internet Traffic	21
2.1.4 Discrete Event Simulation	24
2.2 Pattern Recognition	27
2.2.1 Feature Selection	32
2.3 Intrusion Detection With Emphasis on Flow-based Techniques	36
2.4 Multi Agent Systems With Emphasis on Network Applications	39
2.5 System Intervention Points	40
2.6 Reputation	43
2.7 Evolutionary Computation	46
2.7.1 Evolutionary Algorithms: Concepts and Formalism	47
2.7.2 Evolutionary Algorithms: Software	50
2.8 Self Organization and Emergence	51
2.9 Evaluation of Classification and Heuristic Systems	52
2.10 Summary	54

	Page
III. MFIRE: Network Simulation and Multi Agent System Design	56
3.1 Network Simulation and Multi Agent System Design	
Overview	57
3.2 Formal Problem Specification	60
3.2.1 Classifier Model Selection	63
3.2.2 Spatial Distribution of Agents	64
3.3 Discrete Event Simulation Engine: MASON	65
3.4 Simulated Network Design - MFIRE's Domain of	
Operation	66
3.4.1 Network Simulation Design Objectives	66
3.4.2 Network Simulation Low Level Components	67
3.4.3 Network Simulation Topology	73
3.4.4 Network Simulation Traffic Routing	74
3.4.5 Network Simulation Traffic Design	75
3.4.6 Interfacing with MASON	81
3.5 MFIRE (Multi Agent System) Design	82
3.5.1 MFIRE Design Operational Objectives	84
3.5.2 MFIRE Execution Flow Design	85
3.5.3 MFIRE Robust Communications	90
3.5.4 MFIRE Agent: Classification	91
3.5.5 MFIRE Reputation	95
3.5.6 MFIRE Agent Mobility	97
3.5.7 Agent Distribution: Evidence of Self Organization	98
3.5.8 Stability: An Emergent Property	98
3.5.9 Optimization via a MOEA	99
3.6 Java Implementation	100
3.7 Summary	103
IV. MASNAC: Network Simulation and Multi Agent System	
Design	104
4.1 Overview: Key differences	104
4.2 Network Simulation	104
4.2.1 Node Behavior	105
4.2.2 Size and Scale	106
4.2.3 Topology	106
4.2.4 Packet Payloads	106
4.2.5 Attacks	107
4.3 Multi Agent System	107
4.3.1 MASNAC Classifier	108
4.3.2 MASNAC Communications	108
4.3.3 MASNAC Features	108

	Page
4.3.4 MASNAC Feature Selection	109
4.4 Summary	110
V. Experimentation and Analysis	111
5.1 Experimental Design	111
5.1.1 MASNAC Performance Assessment: Response Variables, Factors, and Statistical Design	113
5.1.2 MFIRE: Qualitative Evaluations	114
5.1.3 Test Computational Environment Factors	115
5.2 Results and Analysis	116
5.2.1 MASNAC Performance Assessment: Results	116
5.2.2 MFIRE and Associated Network Simulation Environment Assessment: Results	121
5.3 Summary	126
VI. Conclusions and Future Research	128
6.1 Conclusions	128
6.2 Future Research Activity	129
6.3 Overall Summary	131
A. Evolutionary Algorithms: Details and Applications	132
1.1 Evolutionary Algorithms: Details	132
1.1.1 Evolutionary Algorithms: Applications	134
B. MFIRE System Details	136
2.1 MFIRE: Messages	136
2.2 MFIRE: Observations	136
C. Test Plan for MFIRE Command, Control, and Communications	140
Bibliography	148
Vita	162

List of Figures

Figure		Page
1	Probability density function for Pareto distribution, $\alpha = 1.0, b = 1.0$	23
2	Basic elements of the MASON model and visualization layers	26
3	A general perspective of a pattern recognition system	27
4	Generic trust model: conceptual relationships	44
5	MFIRE package diagram.	58
6	MFIRE class diagram.	61
7	MFIRE sequence diagram illustrating the transmission of a packet	71
8	Network produced by TopGen and reproduced in MFIRE	74
9	MFIRE: Example block scan conducted by the ScanProcess class	78
10	MFIRE activity and client-server diagrams showing the system's normal flow of execution	86
11	MFIRE detailed activity diagrams for the controller and the agent	89
12	Comparison of reputation curves, using decay vs. not using decay, for single test runs in the MASNAC system	96
13	MFIRE network simulation environment using MASON's visualization and GUI facilities	102
14	Iteration 1 network	106
15	Classification accuracy of MASNAC under different parameters; in each boxplot: no reputation; reputation without decay; reputation with decay	118
16	MFIRE DDoS Attack Test: Console Output	121

Figure	Page
17	MFIRE DDoS Attack Test: Vizualization 122
18	MFIRE: Figure 9 redisplayed to show block scan conducted by the ScanProcess class 123
19	Worm attack growth charts 124
20	Worm attack visualization 125
21	MFIRE: Messages sent by the controller and received by agents 137
22	MFIRE: Messages sent by agents and received by the controller 138
23	MFIRE: Messages sent by agents to other agents 138
24	MFIRE: Messages involved in agent migration 138

List of Tables

Table		Page
1	Characteristics of multi-objective metaheuristics frameworks (from [103])	50
2	How the AgentController Rates Providers of Shared Feature Values	95
3	Comparison of Iterations 1 and 2	105
4	Attack Classes	107
5	Two Agent Classification Accuracy Over 30 Runs	117
6	P Values For Two-Sided Wilcoxon Rank Sum Test for Two Agent System	117
7	Three Agent Classification Accuracy Over 30 Runs	119
8	P Values For Two-Sided Wilcoxon Rank Sum Test for Three Agent System	119
9	Four Agent Classification Accuracy Over 30 Runs	120
10	P Values For Two-Sided Wilcoxon Rank Sum Test for Four Agent System	120

List of Abbreviations

Abbreviation		Page
IDS	Intrusion Detection System	1
AS	Autonomous System	4
IANA	Internet Assigned Numbers Authority	5
BGP	Border Gateway Protocol	5
DDoS	Distributed Denial of Service Attacks	5
DES	Discrete Event Simulation	7
BGP	Border Gateway Protocol	13
ARPANET	Advanced Research Projects Agency Network	15
LAN	Local Area Network	15
WAN	Wide Area Network	15
MAN	Metropolitan Area Network	15
BA	Barabási-Albert	17
NLANR	National Laboratory for Applied Network Research	18
FKP	Fabrikant-Koutsoupias-Papadimitriou	19
CD	Controlled Distance	20
DES	Discrete Event Simulation	22
EPE	Expected Prediction Error	30
SVM	Support Vector Machine	30
MDC	Minimum Distance Classifier	31
NIDS	Network Based IDS	37
HBSS	Host Based Security System	37

Abbreviation	Page
IPFIX	IP Flow Information Export 38
IETC	Internet Engineering Task Force 38
MAS	Multiagent System 39
EA	Evolutionary Algorithm 47
GAs	Genetic Algorithms 47
ESs	Evolution Strategies 47
EP	Evolutionary Programming 47
MCDM	multiple criteria decision making 48
MOEA	Multi Objective Evolutionary Algorithm 49
NSGA-II	Non-Dominated Sorting Genetic Algorithm-II 49
SPEA2	Strength Pareto Evolutionary Algorithm 2 49
RPC	Remote Procedure Calls 59
SOMAS	Self Organized Multi Agent Swarms 65
FIPA	Foundation for Intelligent Physical Agents 66
TCP	Transmission Control Protocol 68
UDP	User Datagram Protocol 68
IANA	Internet Assigned Numbers Authority 68
HTTP	Hyper Text Transfer Protocol 68
BSD	Berkeley Standard Distribution 79
CVE	Common Vulnerabilities and Exposures 79
NTP	Network Time Protocol 90
GUI	Graphical User Interface 102
MOGAs	Multi-Objective Genetic Algorithms 134

Abbreviation		Page
MOMA	Multi Objective Memetic Algorithm	134
PD	projection distance	134
TTL	Time to Live	140

A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION USING REPUTATION AND EVOLUTIONARY COMPUTATION

I. Introduction

Surveying the modern digital expanse of the computer network for entities nefarious and profane is the work of an Intrusion Detection System (IDS). Techniques for intrusion detection are as diverse as their targets, but are often characterized in two ways: 1) network-based or host-based; and 2) signature-based or behavior-based [95].

The signature-based IDS reacts to strings flowing across its aperture that match a signature in a repository of undesirable anomalies. Such a system may be effective for many classes of attacks when caretakers provide a steady supply of up-to-date signatures. As the signature repository grows, however, it is increasingly difficult to compare every packet with every signature. The IDS can become overwhelmed with processing and fail to detect many malicious packets [95]. The behavior-based IDS, on the other hand, reacts to anomalous system activities. Unfortunately, the behavior-based IDS is designed to notice the symptoms of an already-infected subject. A technique complementary to these approaches examines statistics related to the inbound and outbound traffic flows¹ and is particularly suited for attacks that make large disturbances in the distributions of these statistics.

Network-based intrusion detection is typically implemented at the network's gateway. With this eye-in-the-sky vantage point, it may be able to detect patterns involving multiple hosts that individually would appear innocuous. It is technically

¹A flow is a set of messages passing an observation point in the network during a certain time interval. Messages belonging to a flow have a set of common properties, such as source and destination addressing information and a message type [129].

challenging or impossible, however, for the network-based IDS to inspect every packet given today's linespeeds, and will inevitably fail to protect individual hosts from every attack. Host-based intrusion detection can be highly responsive to local situations, but has no appreciation for malicious activity that is inherently distributed across the network. The multi agent system paradigm applied to intrusion detection attempts to get the best of both of these approaches [159]. Autonomous, mobile agents reside transiently at network hosts, and can detect and respond to local problems as well as summarize local information on behalf of a central entity performing network-based intrusion detection functions.

Sperotto et al. [139] survey current flow-based intrusion detection techniques. In each of 14 systems in their survey, data processing is centralized. In all but four of these systems, data *collection* is also centralized. In concluding remarks, they point to the research opportunities in “the development of *distributed flow-based detection* systems” (emphasis in the original).

Hence, this research investigation develops and analyzes two iterations of a multi agent, flow-based intrusion detection system. One of the innovations in this research effort is the use and evaluation of a ‘reputation’ system to govern agent mobility.

1.1 The Generic Intrusion Detection Problem

Effective intrusion detection is vital yet elusive. Despite many good intentions and much effort between a White House Presidential Directive issued in 1990 and another published in 2009, the fundamental vulnerability of electronic systems remains largely unaltered. Such is the premise of [28], which also highlights the admission by the NSA's General Keith Alexander in the Spring of 2010 that even U.S. classified networks have been penetrated [61].

Nation states loom large in the battle for cyberspace, and international tension

is on display. Relatively benign evidence of this is seen in assessments of strategic competitor's network operations capabilities [92]. Of more immediate impact are cases where such capabilities have been put to use. A specific example that recently dominated headlines is Stuxnet. This worm, which gained fame for its disruption of Iran's nuclear production, is widely believed to have been produced by state-funded organizations. It is one of the most sophisticated cyber weapons ever *discovered* [65]. Regarding the most sophisticated cyber weapon ever *developed*, the victims and the public at large may never hear about it.

Mafia-style cyber-crime establishments are also rapidly expanding, giving rise to pervasive phishing and botnet herding activities [6]. Adding further woe to the network security practitioner, even *script kiddies*² remain a threat with the public proliferation of sophisticated scan and attack tools.

Three major classes of network threats include [139]:

1. Attacks that consume network resources, denying their use for legitimate purposes;
2. Attacks that infiltrate systems, allowing attackers unauthorized access to system resources, including sensitive data, data storage, privileged relationships with other systems, and network connectivity;
3. Unauthorized vulnerability scans, providing attackers vital reconnaissance in preparation for infiltrating activities

In truth, these three threats are mutually reinforcing. For example, a successful scan allows an attacker to infiltrate networks with great stealth and precision; once in control of multiple hosts, the attacker may use them to launch a distributed denial

²Script kiddies use tools and techniques developed by others, usually employing them randomly and with little regard or even understanding of the consequences [115].

of service attack on another target system or network. Alternatively, the attacker can use these newly acquired assets to conduct further scans more efficiently / stealthily. As another example, a clever attacker may launch a denial of service attack on a highly visible service to divert the attention of security personnel from his infiltration activities.

These dangers both elevate and implicate the computer network security industry. Concepts such as ‘risk management’ and ‘formal solutions’ make contributions but ultimately fall short of comprehensive defense [156]. At the core of any network attack is simple misuse of technology made attractive by the ease with which anonymity is maintained. To counteract this, perhaps one day network devices will only handle digitally authenticated communications. In the meantime, development of better malicious traffic detection techniques should help the online world cope as best it can.

1.2 Scope of Investigative Domain

One way to cope involves elevating the viewpoint in order to gain a broader perspective of malicious activity traversing and impacting multiple, connected networks. This community of networks perspective shares threat conditions among participants to enable more effective local response [1]. Such a community of networks may arise from a coalition of network owners, or it may fall under the ownership of a single entity, such as a worldwide corporation, government agency, or military.

Therefore, the network of interest to this research is the Autonomous System (AS)-level of the Internet. Each Autonomous System is an administrative domain, comprised of one or more networks of routers, switches, edge devices, and other physical elements, sometimes involving thousands of IP³ addresses. Each Autonomous

³The Internet Protocol specifies the rules for communication across networks at the router level. A message sent using IP is a *packet*.

System is assigned a number by the Internet Assigned Numbers Authority (IANA) for inter-AS addressing. The Border Gateway Protocol (BGP) enables gateway routers in different ASes to advertise the reachability of assigned IP networks and routes to other IP networks.

The typical scenarios developed for our current research involve 100 nodes, each abstractly representing an AS. In the real world, the number of organizations associated with a 100-node network of autonomous systems could number from one up to 100. Large organizations, such as the U.S. Department of Defense, have been assigned well over 100 IPv4⁴ Autonomous System Numbers (though not all are in use at all times).

The lengths of the links connecting the nodes in typical scenarios range from one to ten units, with implications for the geographical area represented. For example, if one allows a single unit link length to represent the length of the connection between two autonomous systems 200 miles apart or less, then the longest link length would be 2,000 miles. The propagation delay⁵ in this scenario ranges from approximately one to ten milliseconds.

Attacks of interest include those that heavily impact the distribution of the traffic arriving at any given node. These are:

- Scans
- Distributed Denial of Service Attacks (DDoS)
- Worms

These are discussed in more detail in Section 3.4.5 in connection with their simulated design and implementation.

⁴Internet Protocol version 4, despite an address space insufficient to meet the demand, still dominates in the networked world of 2011.

⁵Propagation delay measures the time required by the signal to traverse the link. Light travels $3.0 \times 10^8 m/s$.

1.3 Research Goal and Evaluation Hypothesis

Our goal is to develop an effective flow-based, multi agent system for inter-AS network attack classification.

It is our hypothesis that we can increase the effectiveness of a flow-based, multi agent network attack classifier by doing the following:

- Employ reputation to motivate agents to move when perceived as not providing useful information to peers;
- Decay the reputation to provide further impetus for agents to find the best vantage points.

The validation of the hypothesis drives specification of the following research objectives and associated evaluation benchmarks:

1. *Develop an effective network simulation environment appropriate for the problem scope.* We desire to simulate networks of autonomous systems at an appropriate level of abstraction, including network *topology* as well as normal network *traffic*. As a qualitative benchmark, the environment should be complex enough to permit a range of flow-based attacks, and present communications challenges due to propagation delay and link congestion;
2. *Validate the proper functioning of simulated malicious traffic.* For qualitative benchmarks, a DDoS should impede legitimate inter-node communications, a scan should produce a report of potential vulnerabilities on targeted systems, and a worm should demonstrate its ability to non-trivially self-propagate across the network;
3. *Validate the proper command, control, and communications in the multi agent intrusion detection system.* The second design iteration promises actual inter-

process communication in our network simulation as well as a robust communications protocol for the multi agent system. Testing should demonstrate a variety of communications failures and indicate whether the system handles such failures gracefully and is able to resume normal operations autonomously;

4. *Study the effects of several factors on classification accuracy.* Factors include the number of agents, the use of reputation decay, and more generally, whether the reputation system is used at all. Our research development must quantitatively assess the performance of at least one of the iterations of our multi agent system under a variety of parameter values. In particular, we compare the system using reputation to the system not using reputation to assess the impact of reputation under various conditions.

1.4 Investigative Approach

Our research effort consists of two design iterations due to the complex nature of developing ID systems. In each, a network simulation environment, simulation scenarios, and a multi agent system to classify current network activity are developed. Network simulations employ topology and traffic models reflecting associated observations of the Internet. Self-organization in the multi agent systems is promoted via the use of a reputation system and evolutionary algorithms for automatic discovery of effective system parameter settings. The multi agent system in the first iteration has received the benefit of classification performance evaluation as reported in [73, 72]. The multi agent system in the second iteration is qualitatively validated to demonstrate proper functionality of command, control, and communications.

The first investigative design iteration develops:

1. A basic network simulation environment implemented within a Discrete Event Simulation (DES) framework;

2. Scenarios for this network environment involving:
 - (a) 10-node networks;
 - (b) Background traffic patterns mimicking the distribution of traffic seen on the actual Internet;
 - (c) DoS attacks;
3. A multi agent system for network attack classification employing reputation as a means of governing agent mobility.

The second iteration represents an architectural redesign, but incorporates all features from the first iteration and adds many additional features, including:

1. A more complex network simulation environment enabling interprocess communication;
2. 100-node networks with topological characteristics following those of real-world AS networks;
3. Additional malicious scenarios including DDoS attacks, scans and worms;
4. A robust communications protocol, for the multi agent system, designed to handle transmission losses and other errors.

1.4.1 Prospective Beneficiaries.

The envisioned customer of such a system deployed on real networks is a coalition of the organizations associated with the 100 autonomous systems. Members of such a coalition agree to facilitate agent mobility and communications across the AS-network. There are several reasons for accepting this as plausible:

1. In many cases, the number of organizations involved in an AS-network of this size would be small, even including just one organization. These cases greatly simplify the negotiations involved;
2. The agents are designed to collect and share only flow-related statistics, avoiding the sensitivities that come up with sharing signature-based intrusion detection data;
3. Participating organizations may be able to adapt better to conditions observed across the AS-network, for example by changing BGP tables or adjusting traffic filters.

1.4.2 The Use of Reputation.

In support of the fourth research objective, *study the effects of several factors on classification accuracy*, we develop a reputation system. Our reputation system is focused on evaluating agents' ability to share "useful" information. It is used both to determine strength of stochastic preference when agents are selected to share information as well as a trigger for migration to another node.

To elaborate, each agent collects, from its host node, a series of local traffic statistics. Some agents are selected to share summaries of these statistics with other peers. Each agent then makes use of both local as well as shared information to make an individual classification, which is sent to a central entity (controller). Each agent is essentially "voting" on each cycle with respect to the overall classification of the recently observed network activity. The reputation of each agent is affected by whether shared information helped recipients vote in step with the majority, compared with how the recipient would have voted using only local information, resulting in a *rating* in each case. A rating table provides the values used to modify agent reputations for various cases. When an agent's reputation drops below a threshold, the controller

instructs it to move to a different node. Upon moving, reputation is reset to the average of the stationary agents.

Many questions arise with such a design proposal:

- Can this reputation scheme actually increase the system's classification accuracy over time, or does it tend to reinforce collectively bad behavior?
- Can agents coalesce to a single node so as to guarantee all shared information is in agreement with local information, and thus avoid any single agent not voting in step with the majority on account of shared information?
- This being a possibility, can a rating table be designed in such a way that it counteracts this tendency?
- How does one find the values for the rating table leading to the best performance?
- Whether agents share no information and receive no ratings, or share information that never challenges recipients' perspectives and receive neutral ratings, it is possible that some agents can stagnate, performance-wise, over time. Reputations stabilize above the threshold for migration and exploration of the network comes to a halt. If the reputation of each agent is decayed each cycle regardless of received ratings, can this more effectively spur agents to find better vantage points in the network leading to better system performance?
- What is the impact of the number of agents involved?

1.5 Thesis Overview

This chapter frames the problem and explains an approach to solving it. Chapter I delves into the concepts involved in realizing this approach, including flow-based

ID, reputation, and evolutionary computation. Chapter III details the design of the latest iteration of our network simulation and our multi agent system called MFIRE. Chapter IV highlights the differences between this and the first iteration of our network simulation environment and multi agent system called MASNAC. Chapter V presents the experimental performance analysis of MASNAC (first iteration design) as well as the results of system validation for our second iteration efforts, including MFIRE and its associated network simulation environment. Chapter VI concludes with a summary of the research impact and opportunities for future research.

In presenting the second design iteration first (in Chapter III), we avoid traversing twice that ground which is shared in both design iterations between concept and implementation.

In the final analysis, while the multi agent system architecture provides an innovative design demonstrating the integration of ideas with high potential to contribute meaningfully to the evolving ecosystem of intrusion detection systems, the most important contribution is the developed network simulation environment. This environment supports not only the investigation of the subject multi agent system, but may be used for other network research investigations as well where the level of abstraction is suitable for the purpose.

II. Literature Review

Chapter I hinted at the range of threats to the productive use of Cyberspace. Fortunately, there exists a cadre of network security practitioners working unceasingly, evolving tools and techniques capable of beating back the binary barbarians. In this document we propose a new, composite approach that deals with one aspect of flow-based attacks. Specifically, it focuses on the challenge of recognizing 1) that a flow-based attack is taking place; and 2) the essential nature of the attack in order to summon the best response available. An innovative solution is designed and presented in Chapters IV and III. This chapter prepares for that presentation by discussing first the concepts and current research in critical areas germane to the flow-based attack classification system.

One of the foundations in network security research is the modeling of networks of interest. For example, modeling the Internet, that vast and ever-evolving network of networks, is itself the subject of a robust body of research and is discussed in Section 2.1. Pattern recognition techniques are essential to automated identification of hostile network activity and are surveyed in Section 2.2; in particular, classification is discussed as a subtopic of pattern recognition. Section 2.9 comments on the statistical evaluation of classification systems. The application of classification techniques to computer network intrusion detection results in an Intrusion Detection System (IDS). Section 2.3 reviews the features and architecture generally employed in these IDSes. Section 2.4 presents the multi agent system paradigm, which applies distributed artificial intelligence to solve numerous problems, including problems affecting intrusion detection. The next three sections deal with the problem of making a complex system flexible and self-adaptive. Section 2.5 presents a methodology for examining system intervention points to identify where efforts should be most fruitful. Section 2.6 surveys the use of reputation as a means of improving individual and multi agent system

performance in uncertain or even hostile environments. Section 2.7 discusses the ideas and roles of evolutionary computation for identifying parameters that achieve 'good' performance. Finally, the concepts of Self Organization and Emergence and their implications for multi agent systems handling network security functions is presented in Section 2.8.

With an understanding of these critical areas, one can better appreciate the design development in Chapters III and IV.

2.1 Internet Modeling

The Internet is a network with nodes that are themselves networks. Each such network consisting of routers and other networked devices under the same administrative control is called an *Autonomous System* (AS). The routers in each autonomous system that are responsible for forwarding/receiving traffic to/from other autonomous systems are gateway routers. This inter-autonomous system routing is handled via the Border Gateway Protocol (BGP). Though each AS may handle traffic internally in unique ways, all rely on BGP as the glue that binds the Internet together. See [95] for an overview of autonomous systems and BGP.

Clearly it is not appropriate to deploy untested defensive applications that operate at the AS level of the Internet. Testing and evaluation must be carried out via simulation, not only to reduce the operational impact on maintaining high levels of service, but also to provide a variety of controlled conditions for analysis of the system over a range of situations. Naturally we seek a balance between accuracy and efficiency when modeling the Internet. The two principle modeling aspects are *topology* and *traffic*.

See [90] for an illustration of this modeling as well as a testing process for evaluating DDoS countermeasures. We use different tools but a similar logical thought

process of seeking, in the first place, a simulation environment that captures the essential properties of the real-world domain of interest. With such an environment, we should have greater confidence in the performance evaluation of any system proposed for deployment on the real Internet.

2.1.1 Modeling Limitations.

Before surveying some of the ways in which the topology and traffic of the Internet have been modeled, it is important to appreciate the limitations of any model. Inasmuch as there are entities whose qualitative behaviors require representation in the simulation, ultimately one is dealing with *qualitative simulation*. As defined by Kuipers [94], qualitative simulation is the “prediction of the possible behaviors consistent with incomplete knowledge of the structure of the physical system.”

Some of the qualitative behaviors that require representation in our research include the actions of malicious and benign users of the Internet, as well as the engineering decisions giving rise to observed network topologies.

Say and Akin [134] prove that no qualitative simulation is both *sound* (“no trajectory which is the solution of a concrete equation matching the input can be missing from the output”) and *complete* (produces no spurious prediction for any particular input). The problem traces to the inherent incompleteness of mathematics itself¹.

Despite this limitation, repeated refinement of the model allows development of a solution that handles most, if not all, practical input. Our modeling process, demonstrated over the two iterations discussed in Chapters IV and III, is inspired in part by Hildebrandt et al. [75], who advocate starting with a simple model and stepwise refining that model until arriving at the intended real system.

¹Kurt Gödel published proof in 1931 [62] that all formal systems sufficiently powerful to represent addition and multiplication of positive integers and zero (i.e. mathematics) include propositions which cannot be proved or disproved within the system.

With eyes thus opened, it is prudent to proceed to a discussion of the issues involved in Internet AS-level topology and traffic modeling.

2.1.2 Modeling Internet Topology.

In order to create a suitable environment for performance evaluation, one must take into account topological characteristics. As explained by Yook et al. [154], the performance of protocols designed for the Internet is greatly influenced by the network topology. “Protocols that work seamlessly on prototypes fail to scale up, being inefficient on the larger real network.” The real challenge to Internet topology modelers lies in capturing the dynamic characteristics that describe and predict how the topology will grow over time, thus allowing protocols to be evaluated against realistic predictions of the Internet several years into the future.

The early years of Internet topology modeling relied on random graphs ([26], [49]). Waxman’s model ([149]) created graphs probabilistically with respect to the Euclidean distance between nodes, and came to be one of the most popular network models. While it represented small early networks (e.g. ARPANET) successfully, the Internet grew in size and complexity, and the modeling of this “strange beast” needed an overhaul.

Doar recognized this [45] and advocated modeling of different hierarchical levels of networks (e.g. LAN, WAN, MAN). He worked with Calvert et al. to remedy the prevalent use of the Waxman model, which was not intended as a general purpose network topology model, but was designed specifically to compare Minimum Steiner Tree algorithms [30]. The remedy treats intranetwork connectivity separately from internetwork connectivity. Two topology generators resulting from this work are Tiers [45] and Transit-Stub [88], [157].

But it was 1999 that proved to be the watershed year in Internet topology mod-

eling. That year, Faloutsos et al. [52] discovered that the Internet is apparently a scale-free network with a power-law degree distribution. They provide three power-laws that characterize the inter-domain topology measurements derived from BGP routing tables collected in 1998:

- Rank exponent: the outdegree, d_v , of a node v is proportional to the rank of the node, r_v , to the power of a constant, \mathcal{R} :

$$d_v \propto r_v^{\mathcal{R}} \tag{1}$$

The rank r_v of a node v is the resulting sequence index when the nodes are sorted in decreasing order of outdegree. They found typical experimental results for \mathcal{R} to be approximately -0.8 . In other words, they discovered a natural stability to the dominance ratio, in terms of outdegree, of nodes with higher outdegree to nodes with lower outdegree.

- Outdegree exponent: The frequency, f_d , of an outdegree, d , is proportional to the outdegree to the power of a constant, \mathcal{O} :

$$f_d \propto d^{\mathcal{O}} \tag{2}$$

This describes the distribution of the outdegree of Internet nodes. Faloutsos et al. [52] measured values for \mathcal{O} ranging from -2.2 to -2.15 . Lower degrees are more frequent, which observation is quantified by this power-law.

- Eigen exponent: The eigenvalues, λ_i , of a graph are proportional to the order, i , to the power of a constant, ε :

$$\lambda_i \propto i^{\varepsilon} \tag{3}$$

These eigenvalues are calculated using the graph’s adjacency matrix. Faloutsos et al. [52] observed values of -0.47 , -0.50 , and -0.48 , which they deem “practically equal.” Despite the increase in the size of the Internet over 1998, the fact that the eigen exponent remained nearly constant suggests that it captures an essential property of the Internet that characterizes each of their samples.

At about the same time, Barabási and Albert published a paper [14] that came to be hugely influential the burgeoning field of *network science*. They observed power-law relationships for a wide variety of networks in fields ranging from computer science to molecular biology. Their claim: that, “independent of the system and the identity of its constituents, the probability $P(k)$ that a vertex in the network interacts with k other vertices decays as a power-law, following $P(k) \sim k^{-\lambda}$.” This result called for a departure from existing network models to models that incorporate growth and preferential attachment, which came to be known as BA models.

Barabási and Albert (and later Crucitti et al.) then incorporated the Faloutsos results into a study of the attack tolerance of complex networks [4, 40], concluding that because the connectivity of the Internet’s AS-level architecture follows a power-law distribution, that it must be highly resilient to random attacks but vulnerable to attacks targeting select ‘core’ nodes. This is because most of the nodes in such a model have at most a handful of links, while a select few have most of the links.

Barabási later joined Yook to characterize the Internet further and demonstrate the inadequacy of existing topology models [154]. Eventually, BA models started popping up [29, 13, 108, 117, 153]. Efforts to refine the original BA model continue today as power-law-based Internet topology generators proliferate.

This power-law approach is not without its critics. Willinger is particularly strident in his opposition [93, 151]. At issue: whether available measurements and their analysis and modeling efforts support the claims that are made by models based

on power-law node distributions / preferential attachment. Specifically, observe the following problems:

1. Making reliable measurements spanning multiple Internet service providers or autonomous systems is inherently problematic due to a lack of a central authority.
2. Making matters worse is that data sets originally collected for a specific research purpose are then used as primary sources in other research efforts.
3. Compounding this error is the subsequent use of statistical rigor beyond what the quality of available measurements justifies.
4. Finally, there is a problem with how model validation in networking research typically happens. Having arrived at a set of statistics that are already based on dubious sets of measurements, any models that demonstrate consistency with these statistics are declared valid. The problem is that, in many cases, alternative models with a wide range of structure can also be produced with the same level of consistency, and no criteria is used to rule them out. Furthermore, it is rarely if ever the case that an independently-compiled dataset and a separate set of statistics are used for model validation.

Willinger’s opposition [93] to the Faloutsos brothers’ conclusions [52] with respect to the AS-level Internet topology involves all of these things. The datasets used in [52] came from *The National Laboratory for Applied Network Research (NLNR)*, which constructed inferred AS connectivity maps by relying on full BGP routing tables collected by the *Route Views Project at the University of Oregon*. The purpose of this project: “to respond to interest on the part of operators in determining how the global routing system viewed their prefixes and/or AS space”. Yet ever since [52], the

resulting datasets have been used to infer Internet AS-level topology. Whether this is legitimate or not is impossible to decide given the lack of any relevant metadata for these datasets.

At best, all that can be concluded from the BGP-derived AS maps are “Pareto-type principles; that is, a small number of nodes have many neighbors, while most nodes are connected to only a small number of neighbors” [93].

What is advocated is that instead of using data fitting as the primary driver of model selection and validation, one should “rely on domain knowledge and exploit the details that matter when dealing with a highly engineered system such as the Internet” [151].

An interesting intersection of this proposed approach and the field’s persisting enthusiasm for power-law models is found in [50]. The authors successfully generate topologies that exhibit the power-law relationships found in [52], but instead of the common, purely stochastic approach, these properties arise from a simple multi-objective optimization, involving “last mile” connection costs and transmission delays as measured in hops.

This model is known as *Fabrikant-Koutsoupias-Papadimitriou (FKP)* and captures these two objectives in the following way. From [138], each node i arrives at a uniformly random point and attaches itself to the node j that minimizes the weighted sum

$$\min_{j < i} \{ \alpha \cdot d_{ij} + ecc(j) \} \tag{4}$$

In this equation, d_{ij} is the Euclidean distance between the nodes and represents the “last mile cost.” The relative importance of this objective is controlled via the weight α . The second term is the *eccentricity* of j and captures the distance from j to the center.

Spatharis et al. present an updated treatment of FKP with several extensions [138]. One extension is the Controlled Distance (CD) Model. The goal of this extension is to address the need for edges between nodes that are not quite leaves, nor particularly central, but are of intermediate centrality. As each node i is added to the network and linked to the node j according to equation 4, a second edge is attached from j to another node k minimizing

$$\min_k \{ \alpha \cdot d_{jk} + ecc(k) \} \quad (5)$$

over all k such that the hop distance from j to k is at most a constant c .

This model decreases the power law exponent while having high average degree and several leaves. The authors of [138] declare this to be, in many ways, the “best performing” of their models in achieving similarity to the Internet’s AS graph. This model and various alternatives are packaged by the authors in the package TopGen.

Other topology generators include:

- Tiers [45]
- GT-ITM - Georgia Tech Internetwork Topology Models [30]
- Inet [153]
- nem [108]
- BRITE [117]
- GDTANG - Geographic Directed Tel Aviv University Network Generator [13]
- RealNet [37], [36]

RealNet is one of the more recent additions to this list. It relies on publicly available datasets including BGP tables and traceroute records, as did [52], but ad-

dresses some of the problems inherent in these datasets and does not attempt to fit specific power-law-based statistics. For example, it gives direct consideration to the IP-aliasing problem, whereby more routers may be inferred than actually exist because each router has a different IP address for each of its interfaces. It also factors in likely policy relationships between neighboring autonomous systems.

In summary, while it is agreed that the AS-level topology of the Internet exhibits Pareto-type principles, the power-law approach to modeling the AS-level topology growth is by itself inadequate. An approach involving domain knowledge and an engineering mindset is preferred. The ideal topology modeling approach will appease both 1) the engineers responsible for implementing actual Internet topologies, as well as 2) the protocol and application developers seeking validation that their efforts will perform adequately on the Internet for at least several years. Currently, this ideal approach is elusive. In the meantime, FKP provides a reasonable balance; RealNet is intriguing but not available for the current research.

2.1.3 Modeling Internet Traffic.

Just as important as modeling topology is the modeling of the traffic that rides on it. The history of traffic modeling begins with Poisson distributions, and ends up with models that exhibit self-similarity [19]. As has been conjectured for the Internet's topology, the Internet's traffic has actually been shown with more statistical rigor to have certain scale-invariant statistics. The common adjective applied to Internet traffic is "bursty" - sharp peaks are observed in the volume of traffic no matter the time scale, which is not the case for Poisson distributions. One can set the rate λ of a Poisson process to emulate burstiness at a target time scale, but as the time scale is increased while λ is held constant, the peaks and troughs in activity flatten until they disappear altogether [152].

Frost and Melamed [57] provide some formalism geared for a Discrete Event Simulation (DES) environment, in which events are modeled as happening at discrete time steps [12]. At a particular node, traffic is modeled as arriving in a sequence of arrival instants $T_1, T_2, \dots, T_n, \dots$. Equivalent representations include *counting processes* and *interarrival time processes*. A counting process is of the form $\{N(t)\}_{t=0}^\infty$, where $N(t) = \max\{n : T_n \leq t\}$ is the number of traffic arrivals in the interval $(0, t]$. For the interarrival time process, $\{A_n\}_{n=1}^\infty$, where $A_n = T_n - T_{n-1}$ is the length of the time interval between the $(n - 1)$ th and the n th arrivals. Equivalence of these representations is:

$$\{N(t) \equiv n\} \equiv \{T_n \leq t < T_{n+1}\} = \left\{ \sum_{k=1}^n A_k \leq t < \sum_{k=1}^{n+1} A_k \right\}$$

since $T_n = \sum_{k=1}^n A_k$.

The amount of traffic arriving at the node at T_n is modeled as B_n , which is a member of the non-negative random sequence $\{B_n\}_{n=1}^\infty$, stochastically independent from $\{A_n\}$.

It is commonly the case in today's context of high-bandwidth networks (as compared to 1994) that, across most nodes, one can assume a constant stream of traffic, particularly for the nodes of the AS-level graph of the Internet. In other words, A_n is usually 1. Let us consider how this situation may arise naturally from Frost and Melamed's formulation. If $\{A_n\}$ is generated for one timescale, then there is a "zoom out factor" for which one could generate $\{A'_m\}$ such that $P(A'_m = 1)$ is arbitrarily high, and it may be more useful to focus purely on the generation of $\{B'_m\}$.

Let T_i represent an arrival time in the original time scale, and T'_j represent an arrival time in a time scale zoomed out by a factor of λ . To simplify matters, we allow $B'_j = 0$. Then $T'_j = j$, and

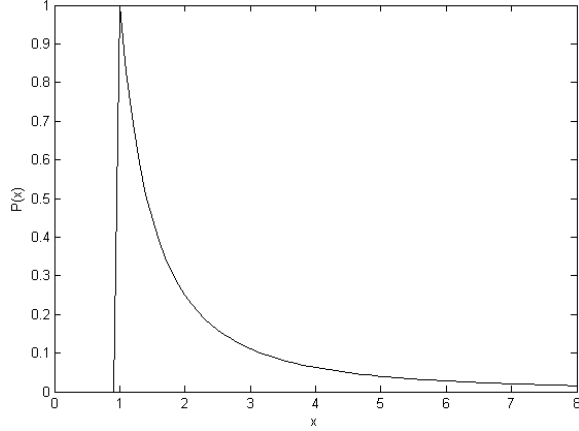


Figure 1. Probability density function for Pareto distribution, $\alpha = 1.0$, $b = 1.0$

$$B'_j = \sum_{i=\lambda j}^{(\lambda+1)j} B_i$$

Frost and Melamed focus on the $\{A_n\}$ representation and use it to describe a broad collection of traffic generation models, with varying emphasis on demonstrating autocorrelation. This matters because of their explanation that “strong positive autocorrelations are a particularly major cause of burstiness.” Since bursty traffic is expected to dominate broadband networks, “models that capture the autocorrelated nature of traffic are essential for predicting the performance of emerging broadband networks” [57].

The Poisson model does not do this, and was only applied to Internet traffic in the first place because it had enjoyed decades of success describing telephonic traffic [152]. Ironically, this is akin to the misadventures in topology modeling described in Section 2.1.2, where datasets collected with one purpose in mind were commandeered for another. In [152], Willinger and Paxson are advocates of turning to better, fractal-like traffic distribution models. This is the same Willinger that today decries the use of power-laws to model Internet topology (see Section 2.1.2). It turns out that for traffic, the case for scale-invariance is much more justified.

The Pareto model exhibits scale-invariant behavior [63]. It has a density function $P(X) = \frac{ab^\alpha}{x^{\alpha+1}}$, $x \geq b$, which has a heavy tail [90]. Figure 1 shows an example for $\alpha = 1.0$, $b = 1.0$. Willinger and Paxson explain that this heavy tail accounts for the fractal nature of aggregated network traffic [152]. To generate a random Pareto-distributed sample, inverse transform sampling is used. Given a random variable U drawn from the uniform distribution $(0, 1)$, T given by

$$T = \frac{b}{U^{\frac{1}{\alpha}}} \tag{6}$$

is Pareto-distributed [44].

This concludes discussion of network topology and traffic modeling. One could also consider models that generate *workloads*, which may be appropriate if the effects of traffic on servers are considered. An example effort to generate representative web workloads for network and server performance evaluation is presented in [15]. Also, one could model the spread of malicious traffic. For example, many models have been built for worm propagations, [35, 104, 147, 163, 164, 84, 135].

In any case, given a model, the next question to consider is how to implement the model for simulation. This is consequently the subject of the next section.

2.1.4 Discrete Event Simulation.

The preceding discussion illustrated the challenges particular to modeling the Internet’s topology and traffic, for the purpose of producing a “useable” simulation. By “useable”, we mean that applications tested under such a simulation generate behaviors tolerably isomorphic to what is seen in a real-world operational environment.

This section focuses on the underlying simulation framework. Specifically, this section is about Discrete Event Simulation.

This method views the simulation as being composed of a chronological sequence of

events, each of which occurs in an instant and changes the state in the system, possibly resulting in more events being scheduled. Comprehensive treatment of Discrete Event Simulation is given in [12].

Components of DES systems include:

- Clock - The simulation keeps track of current simulation time in appropriate measurement units, but unlike in real time simulations, time in a DES jumps from one instantaneous event to the next.
- Schedule - The set of events to handle, typically implemented as a priority queue sorted by event time.
- Random-Number Generator - pseudorandom, which is desired in order to support a rerun of a simulation with exactly the same behavior

Typical usage of a DES includes the gathering of statistics, for which facilities may be provided, and the specification of a stopping condition. As may be the case with continuous- but not real-time simulation, a discrete event simulation runs at a rate that is not tied to the real-world clock. When resources permit, simulations may be run potentially much faster than real time, which is useful for collecting large amounts of statistics. In other cases, it may be desired that simulations run much slower than real time, perhaps paused for an extensive period of time via checkpointing, which is useful for direct observation and analysis of system dynamics.

Parallelization of DES is discussed extensively in [58]. More recently, Park and Fishwick present their work using graphics processing unit-based clusters in [125].

2.1.4.1 Popular DES Engines.

Some of the more well-known DES options and their areas of emphasis are:

- OMNeT++, [145]: network simulation

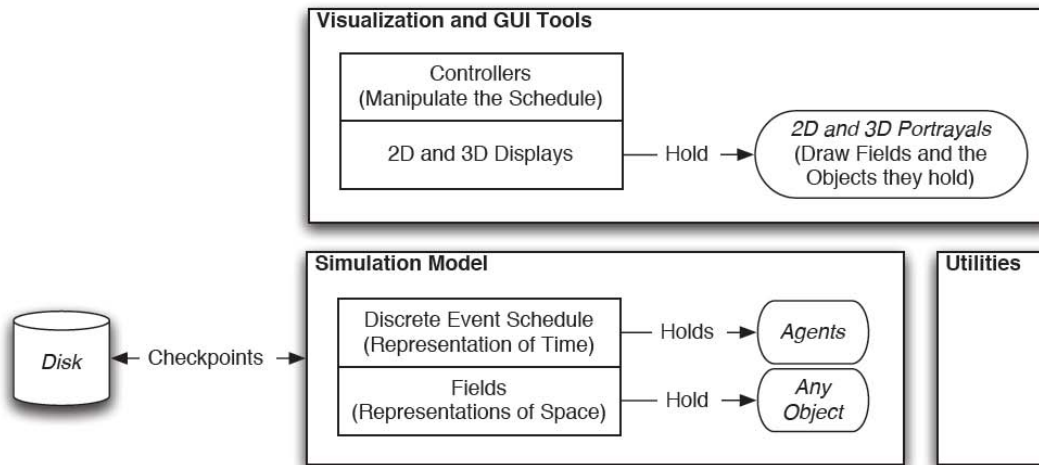


Figure 2. Basic elements of the MASON model and visualization layers

- MASON, [106]: agent-based systems simulation
- CNET, [114, 113]: network simulation
- GloMoSim, [158]: large-scale wireless networks
- ns2, [112]: network simulation
- PARSEC, [11]: parallelization

2.1.4.2 MASON Overview.

MASON, developed at George Mason University and presented in [106], is ‘a single-process DES core and visualization toolkit written in Java’. It is flexible enough that it can be used for a wide range of simulations, but emphasizes support for “swarm” simulations with up to millions of agents. It is fast and portable and produces guaranteed replicable results, courtesy of checkpointing facilities.

The underlying model runs in a layer independent of the visualization layer. Thus, while the visualization facilities enable easy interaction with simulations, simulations may run without visualization, or the visualization can be changed at will, perhaps

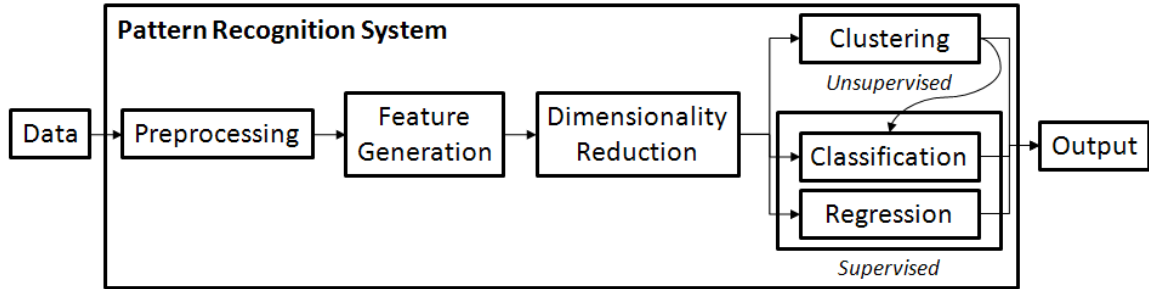


Figure 3. A general perspective of a pattern recognition system

according to the preferences of the observer. The basic elements of the MASON model and visualization layers are presented in Figure 2.

MASON has been used successfully for applications ranging from physics demonstrations to cooperative target observation in unmanned aerial vehicles to the testing of ant foraging algorithms.

2.2 Pattern Recognition

The preceding sections have laid the groundwork for a simulation framework that will adequately reflect the contested domain of the Internet. Ultimately, the purpose of the present research is to automate the detection and classification of malicious network activity. If the classification is accurate, precise, and timely, the malicious activity may be counteracted.

The fact that legitimate traffic has been characterized as ‘bursty’ lends confidence to the idea that deviations from the emergent features of legitimate, bursty internet traffic may reveal the presence of anomalous (and potentially malicious) traffic, using techniques from the broad field of *pattern recognition*.

Authoritative texts on pattern recognition include [46], [74], and [24]. A general description of a pattern recognition system includes the elements depicted in Fig. 3. We generally rely on the notation found in [74] in the following discussion.

Preprocessing formats the data, possibly performing some filtering in the case of

noise or some system or environmental anomaly.

Feature generation, sometimes referred to as *feature extraction*, is the transformation of raw data into derived data points that may facilitate the characterization of the observed process. One can even use the raw data itself for feature measurements. Commonly, features are statistical measurements of the raw data or may be the result of passing the raw data through a mathematical transformation (e.g. Fourier coefficients of a signal or wavelet coefficients of some image). The feature or vector of features is represented as X . If X is a vector, it has p elements, and components are accessed via subscripts X_j .

Dimensionality reduction, or *feature selection*, filters the available features with the premise that not all features are useful. In fact, some features may even be harmful (misleading). Even if all features are useful, resource limitations, in terms of computation, bandwidth, or storage, may require filtering the least beneficial information prior to performing clustering, classification, or regression. The feature selection vector may be represented as $\vec{\sigma} \in \{0, 1\}^p$.

Clustering, classification, and regression represent the three fundamental problems of pattern recognition, one or more of which must be addressed by any pattern recognition system.

Clustering seeks to identify the natural groupings of the data. This effort can involve considerable subjectivity. Typically, the number of groupings or *clusters* is not known beforehand. The solution is to either simply specify the desired number of clusters and evaluate the resulting cluster assignments, or define some distance-based threshold from which the number of clusters is derived.

In clustering, we typically require a measure of dissimilarity $d_j(x_{ij}, x_{i'j})$ between

values of the j th attribute. Then

$$D(x_i, x_{i'}) \triangleq \sum_{j=1}^p w_j \cdot d_j(x_{ij}, x_{i'j}); \sum_{j=1}^p w_j = 1.$$

is the dissimilarity between objects i and i' given the inputs $x_i, x_{i'}$ and weight vector w . Usually, $d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$, but other choices are possible, or even required in the case of nonquantitative attributes [74].

Clustering is the search for an encoder $C(i)$ that assigns the i th of N observations to one of K clusters. An encoder may be evaluated by measuring the *between-class scatter* to *within-class scatter* ratio, $\frac{B(C)}{W(C)}$. Between-class scatter is defined as

$$B(C) \triangleq \frac{1}{2} \sum_{k=1}^K \sum_{C(i)=k} \sum_{C(i') \neq k} d(x_i, x_{i'})$$

while within-class scatter is

$$W(C) \triangleq \frac{1}{2} \sum_{k=1}^K \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$$

We desire high $\frac{B(C)}{W(C)}$ in order to achieve the goal of high between-class scatter and low within-class scatter, thus establishing clusters with well-defined boundaries [74].

Classification is a process that assigns one of a discrete number of labels to each data point (input vector). A is the ‘true’ output and takes values from the set \mathcal{A} . The classifier is \hat{A} and should also take values from \mathcal{A} . *Regression* seeks to model some continuous process (function). The output of the function being modeled is denoted Y and takes values from some continuous set, such as \mathbb{R} , and a predictor for Y is \hat{Y} [74].

Unlike the unsupervised learning technique of clustering, classification and regression as supervised learning techniques require training data in which inputs are

associated with known output (e.g. the ‘correct’ label or value corresponding to each sample in the training data). Based on the specific classification or regression technique selected by the system designer, the system derives the necessary parameter values for a process that reliably transforms the training input into the desired output. This notion is formalized as the minimization of Expected Prediction Error (EPE):

$$\text{EPE} \triangleq \text{E}[L(A, \hat{A}(X))] \tag{7}$$

L is a *loss* function, and expectation is taken with respect to the joint distribution $P(A, X)$. With $K = |\mathcal{A}|$ classes, the loss function may be represented as a $K \times K$ matrix \mathbf{L} . This loss matrix has values of zero on the diagonal. Everywhere else, a non-zero value $L(k, l)$ indicates the penalty for misclassifying an observation as belonging to \mathcal{A}_l when it actually belonged to \mathcal{A}_k .

Hastie et al. [74] show that by conditioning on X , we can rewrite 7:

$$\text{EPE} = \text{E}_X \sum_{k=1}^K L[\mathcal{A}_k, \hat{A}(X)]P(\mathcal{A}_k|X) \tag{8}$$

When the loss function is *zero-one*, meaning that a single unit penalty is assessed for any misclassification, the intuitive guidance for $\hat{A}(X)$ is:

$$\hat{A}(X) = \mathcal{A}_k \text{ if } P(\mathcal{A}_k|X = x) = \max_{a \in \mathcal{A}} P(a|X = x) \tag{9}$$

In other words, the classification output should be the most probable class given the input (*Bayes classifier*). Naturally, what makes this difficult is the fact that one has to estimate the probabilities using a limited set of data.

Kotsiantis reviews several of the more popular classification techniques [91]. One of these is the Support Vector Machine (SVM), which is used in this research. For

mathematical details, the reader is referred to [91, 143] and [74], but the idea is to find the hyperplane that separates the training data with the maximum margin - the distance on either side to the nearest samples.

The SVM is preferred in our current research due to its high generalization performance without the need to add *a priori* knowledge, even in the presence of many features in the input space [34].

The first iteration of our research employs a simpler technique known as the Minimum Distance Classifier (MDC) [56]:

$$\hat{A}(x) = \min_{a_k \in \mathcal{A}} \|x - a'_k\| \quad (10)$$

where a'_k is the *prototype* or *sample class mean* of class \mathcal{A}_k . That is, for $1 \leq j \leq p$, and given a sample set X , a'_{kj} is the arithmetic mean of values x_j for all $x \in X | A(x) = \mathcal{A}_k$.

The MDC technique is very simple and works well when the distance between class means is greater than the typical variance within each class [56].

Regardless of the technique, one of the issues confronting the designer of a pattern recognition system is how to avoid *overfitting*, or ‘fitting to the noise.’ The wrong approach to training the system can result in a system that performs well on the training data but performs poorly on other data sets, demonstrating a lack of general applicability. Two contributing aspects of this are *bias* and *variance*. Bias marks the difference between the sample mean and the true mean, and variance is simply the measure of variability in the sample set. Large training and validation sets that accurately represent the target domain can address all of these issues, but obtaining sufficient data may be prohibitively expensive or otherwise impossible. Thus, ingenuity in compensatory techniques may be required if it is desired to accurately predict the error of a proposed classifier.

One such compensation technique is *K-fold cross-validation*. This approach splits

the data into K roughly equal-sized parts. For each of K rounds, one of these parts is set aside for validation while the rest are involved in training or fitting the model. The errors from all rounds are combined for the final cross-validation estimate of the prediction error. Specifically,

$$CV(\hat{a}) \triangleq \frac{1}{N} \sum_{i=1}^N L(a_i, \hat{a}^{-\kappa(i)}(x_i))$$

where \hat{a} is the fitted classifier, N is the number of samples available, L is the loss function as before, and κ is an indexing function that indicates the partition to which a given observation is allocated for the purposes of this cross-validation technique. The term $\hat{a}^{-k}(x)$ denotes the fitted classifier trained with all but the k th partition of the data [74].

The choice of K is reflective of the bias-variance tradeoff that is so common in pattern recognition: higher K (maximum value is N) yields lower bias for the true prediction error, but higher variance. Common values of K are five and ten [74].

If one wishes simply to determine the optimum ratio of training set to validation set, Guyon gives the following guideline [67]: let the ratio of the validation set size over the training set size scale like the square root of the complexity of the second level of inference (minimizing the validation error) over the complexity of the second level of inference (minimizing the error rate on the training set).

2.2.1 Feature Selection.

As noted by Gates et al. in [66], the three principle objectives of feature selection are: 1) improving prediction performance; 2) enabling faster, more efficient prediction; and 3) providing a better understanding of the underlying process that generated the data.

One of the primary reasons feature selection has the potential to greatly improve

prediction performance is that it directly confronts the *curse of dimensionality* [22]. Hastie et al. [74] examine some of the many manifestations of this problem. Essentially, such manifestations arise from the fact that in order to maintain the same sampling density enjoyed in a lower dimension, the number of samples must increase exponentially as one moves to higher dimensions. Usually, the number of samples practically attainable is far fewer than necessary to maintain the desired sampling density. Some of the consequences of sparse sampling in high dimensions are:

- Techniques that rely on information gleaned from local neighborhoods (e.g. k -nearest neighbors) break down. In low dimensions, it may be possible to form local neighborhoods that cover some percentage of the sample population. But if one desires to cover the same percentage in high dimensions, the neighborhood ends up covering most of the range of each input variable and is no longer local. If instead the imperative is to strive for truly local neighborhoods in high dimensions, the number of samples involved in the characterization of each neighborhood shrinks and variance jumps.
- All sample points are close to an edge of the sample space. This makes interpolation impossible. One must extrapolate from neighboring sample points, which results in predictions with much greater uncertainty.

These issues motivate a focus on a reasonably-sized subset of the available features. Finding the *optimal* subset, however, is known to be NP-hard [5]. Consequently, feature selection techniques typically involve heuristics that lead to “good” quality feature subsets in a reasonable amount of time (i.e. polynomial in the number of features).

Feature selection techniques are typically categorized as *filter*, *wrapper*, or *embedded methods*, discussed respectively in Sections 2.2.1.1, 2.2.1.2, and 2.2.1.3.

2.2.1.1 Filter Methods.

A filter method typically involves some notion of *feature ranking* independent of the choice of the predictor. This is computationally efficient because it requires only the computation of p scores and sorting the scores. It introduces bias but may have considerably less variance compared to other methods and is therefore robust against overfitting [66, 74]. Feature ranking methods include analyzing performance as a single variable classifier and information theoretic ranking criteria.

These filter techniques can be useful but also incur limitations. The underlying assumption is that variable dependencies can be ignored, but in practice, this is not always the case. Guyon and Elisseeff [66] provide examples in response to three probing questions:

1. Can presumably redundant variables help each other? (yes; independently and identically distributed variables are not truly redundant)
2. How does correlation impact variable redundancy? (Perfectly correlated variables are truly redundant and provide no performance gain, but very high variable correlation or anti-correlation does not mean absence of variable complementarity.)
3. Can a variable that is useless by itself be useful with others? (Yes. In fact, even two variables that are useless by themselves may be very useful together.)

Methods that score variables individually and independently of each other are at a loss to determine which combination of variables would give the best performance. Nevertheless, for combining computational efficiency with reasonable performance, as well as general applicability across a range of classification algorithms, filter methods can work well in practice.

A common technique for filtering features involves ranking them according to how well they separate sample distributions collected from two classes. The Bhattacharyya distance [23], named after the mathematician, is a measure of the distance between two sample distributions. If two sets of samples are produced by the same process, the estimated distributions should be very close, and the Bhattacharyya distance near zero. Formally, for discrete probability distributions p and q over the same domain X , the estimate of the Bhattacharyya distance $D_B(p, q)$ is:

$$D_B(p, q) \triangleq -\ln(BC(p, q)) \quad (11)$$

where

$$BC(p, q) \triangleq \sum_{x \in X} \sqrt{p(x)q(x)} \quad (12)$$

is the *Bhattacharyya coefficient*. Simply, for each value of $x \in X$ found in both sample sets p and q , $BC(p, q)$ increases, up to a maximum of 1 when $p(x) = q(x)$ for all $x \in X$. In this case, $D_B(p, q) = 0$. Conversely, as the limit of $BC(p, q)$ approaches 0, observe that $D_B(p, q)$ increases without bound.

2.2.1.2 Wrapper Methods.

Wrapper methods test feature subsets against the chosen learning machine, which is regarded as a black box. Questions include 1) how to search the space of all possible variable subsets; 2) how to assess the prediction performance of a learning machine to guide the search and halt it; and 3) which learning machine or predictor to use (see [89]).

2.2.1.3 Embedded Methods.

Embedded methods contrast with both filter methods, which employ no learning whatsoever, and wrapper methods, which are general techniques that may be applied to any classifier. In embedded methods, the structure of the class of functions under consideration plays a crucial role, and techniques are developed that are specific to certain classifiers. See [101].

2.3 Intrusion Detection With Emphasis on Flow-based Techniques

The previous section introduced and surveyed the field of pattern recognition, including classification systems in particular. Our research applies classification techniques to the detection and characterization of malicious activity on a simulated subset of the AS-level Internet.

Recognizing malicious activity on a computer network is called Intrusion Detection, and it is the job of the Intrusion Detection System (IDS). An IDS is characterized by the techniques employed to accomplish this task. Axelsson provides an IDS survey and taxonomy [7], which is the basis of some of the following discussion.

A *semantic*-based IDS examines packets for strings matching signatures of previously identified malicious activity. A *flow*-based IDS focuses attention on the sizes and frequency of packets associated with source-destination communication sessions to detect whether the communication pattern is evidence of malicious activity. A *behavior*-based IDS observes potential victim devices for deviations from ‘normal’ behavior, which is taken as evidence of an attack.

Another way to characterize an IDS is according to whether it employs *self-learning* or is *programmed*, or some combination thereof. Other IDS characteristics include:

- real- or near real-time detection / non-real-time detection

- process data continuously / process data in batches at regular intervals
- rely on network data - Network Based IDS (NIDS) / rely on host-based security logs - Host Based Security System (HBSS)
- passive response / active response
- process data centrally / process data distributively

Axelsson defines the *effectiveness* of an IDS as the degree to which it can correctly classify intrusions and avoid false alarms. In the simple binary case, where one simply needs to know whether a traffic sample is malicious or benign, the classification process can be analyzed in terms of its performance in four measures [121]:

- probability of declaring the sample malicious when it is, in fact, malicious (*true positive*);
- probability of declaring the sample benign when it is truly benign (*true negative*);
- probability of declaring the sample benign when it is actually malicious (*Type I error* or *false negative*);
- probability of declaring the sample malicious when it is really benign (a *Type II error* or *false positive*).

In both semantic- and flow-based approaches, packets may be *sampled* when resources cannot keep up with heavy volume and fast connections. The impact of this lossy sampling is analyzed in [27, 109, 165].

Our research focuses on flow-based intrusion detection. For an overview of this approach, see Sperotto et al. [139]. The preferred definition of IP flow comes from the

IP Flow Information Export (IPFIX) working group within the Internet Engineering Task Force (IETF) [129]:

A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties.

These common properties are called *flow keys*. Typically, they are: source and destination IP addresses, source and destination port numbers, and the IP protocol [139].

Sperotto asserts that flow-based intrusion detection is appropriate for dealing with flow-based attacks, such as:

- Scans
- DDoS
- Worms
- Botnets

Notable applications of this detection technique include:

- a decision tree-based abnormal traffic detection method that operates on flows [85]
- scan detection using logistic regression modeling [60]
- worm detection using entropy [148]
- real-time analysis of flow data in TOPAS [124]
- high-speed flow-based intrusion detection based on *sketches* [59]

- detection of super sources and destinations in high-speed networks [160]
- network-wide anomalies in traffic flows [100, 98, 99, 106].

2.4 Multi Agent Systems With Emphasis on Network Applications

A common design question for any IDS is how to maximize the benefits and minimize the penalties associated with network-based as well as host-based approaches. The Multiagent System (MAS) paradigm offers a way to accomplish this, with the added advantages of flexibility and robustness provided by this approach.

What is a multiagent system? First, what is an agent? Russell and Norvig [133] require several properties: autonomous operation, ability to perceive the environment, persistence over a long period of time, ability to adapt to change, and ability to create and pursue goals. These goals are typically in support of a broader objective. Franklin and Graesser [55] provide a survey of definitions for software agents, and an associated taxonomy.

If one agent is good, in many cases more are better. A multiagent system, naturally, is a collection of agents that collaborate, explicitly (e.g. via cooperation) or implicitly (e.g. via competition) to achieve a broad objective or series of objectives.

In a MAS with *mobile* agents, all hosts in the network must have a generic agent platform installed which provides the environment in which the agent executes. Agent migration then consists of sending agent state to a remote process responsible for reinstantiating the agent.

Jansen lists some specific advantages of a mobile, agent-based IDS [82]:

- *Overcoming network latency* - if an agent is present on a node requiring remedial action, the agent can respond more quickly than if action must be initiated by a central coordinator

- *Reducing network load* - Communication requirements are reduced by allowing agents to process sensor data locally, instead of requiring each node to send sets of sensor observations to a central processing location. Sharing the results of local processing incurs a relatively light demand on bandwidth.
- *Autonomous execution* - surviving agents continue to operate when part of the IDS fails
- *Platform independence* - agent platforms with standard interfaces may be written for multiple operating systems to allow effective MAS execution in a heterogeneous OS environment
- *Dynamic adaptation* - the system can be reconfigured during run-time in a variety of ways. The mobility of the agents allowing them to seek effective positions is a reconfiguration. Agents can clone themselves or request assistance from other agents in high demand situations. Selected agents can be replaced while non-selected agents continue to operate. One can also update repositories of behaviors and parameters which agents access periodically.

Potential disadvantages include decreased performance and/or increased resource consumption when mobility is implemented ineffectively. Also, since each agent is a member of a trusted network that, if compromised, could provide the attacker considerable leverage, digitally signed communications (including migrations) are essential.

2.5 System Intervention Points

The preceding section provided an overview of multi agent system. This section examines potential system intervention points to consider when setting up such a system. Indeed, any multi agent system for network intrusion detection is invariably

complex. Configuring the system's parameters in order to maximize effectiveness and efficiency is difficult but critical.

Meadows offers a list of intervention or leverage points that may represent opportunities for change when considering how to improve system performance [116]. By *system*, she refers to the generic definition as opposed to the multi agent systems discussed in the previous section. Meadows' research is concerned with large and sometimes abstract systems, including corporations, economies, living organisms, cities, and ecosystems. Regardless of the type of system with which one is specifically concerned, if it has any complexity to it at all, certain abstract notions from the field of systems analysis may be applied to guide investigations into the system. The purpose of such investigations is to understand and/or change the system in some way. They are "guided" in the sense that one seeks to arrive at some level of understanding or achieve some effect in an efficient way. To determine where to intervene in a system for greatest effect, Meadows lists the potential leverage points in increasing order of effectiveness [116]:

12. Constants, parameters, numbers (e.g. subsidies, taxes, standards)
11. The sizes of buffers and other stabilizing stocks, relative to their flows
10. The structure of material stocks and flows (such as transport networks, population age structures)
9. The lengths of delays, relative to the rate of system change
8. The strength of negative feedback loops, relative to the impacts they are trying to correct against
7. The gain around driving positive feedback loops

6. The structure of information flows (who does and does not have access to what kinds of information)
5. The rules of the system (such as incentives, punishments, constraints)
4. The power to add, change, evolve, or self-organize system structure
3. The goals of the system
2. The mindset or paradigm out of which the system—its goals, structure, rules, delays, parameters—arises
1. The power to transcend paradigms

For a detailed explanation of the terms in the list, see the original paper [116]. Clearly, it may not be possible to change all of the items in this list. It is generally beyond the scope of a software system design to transcend paradigms (item 1) or change the initial paradigm (item 2) in which the design task was conceived. Furthermore, the software system designer may or may not be able to influence or refine the requirements of the system. But many of the remaining elements of list comprise aspects of the system design largely under the designer’s direct control.

Of special interest is the concept of *self-organization*. It is the ability of the system to change itself by changing anything lower on the list. It endows the system with resilience, which is particularly important in the domain of quickly-evolving cyber threats. “Self-organization,” says Meadows, “is basically the combination of evolutionary raw material—a highly variable stock of information from which to select possible patterns—and a means for experimentation, for selecting and testing new patterns” [116]. In other words, the system has the capacity to adapt in pursuit of better performance (or survival in the face of deteriorating conditions). For other definitions of self-organization, see [41, 71, 128] and Section 2.8.

Sections 2.6 and 2.7 present two ideas that can be incorporated into a MAS in order to achieve this coveted prize of self-organization. Section 2.6 explores the use of *reputation* to guide agent collaboration and migration. Section 2.7 details the use of evolutionary computation, which more explicitly models biological adaptation in the wild via notions of population selection based on fitness, offspring production with genetic crossover and mutation, and so forth. Section 2.8 distinguishes between self-organization and emergence, and discusses both concepts more fully in the context of multi agent systems.

2.6 Reputation

The pursuit of an adaptable multi agent system achieving Meadows' notion of self organization leads to the need for a way to effectively govern the agents' communication and mobility patterns.

As introduced in Section 1.4.2, we consider the use of *reputation* to achieve this purpose. Reputation is defined simply as the collective observation, by a society, of a particular agent's past behavior. A reputation system provides publicly-available assessments of agents' trustworthiness based on ratings from past transactions [136].

Trust, on the other hand, is a subjective (internal) measure by which a particular agent makes use of reputation and/or records of direct experience to govern its interactions with other agents [80].

A variety of trust models exist. Huynh et al. [80] review three distinct modeling approaches:

- Mechanisms deriving trust via certificates, rules, and policies
- Centralized trust mechanisms in which witness observations are collected by a central authority; also known as centralized *reputation* mechanisms

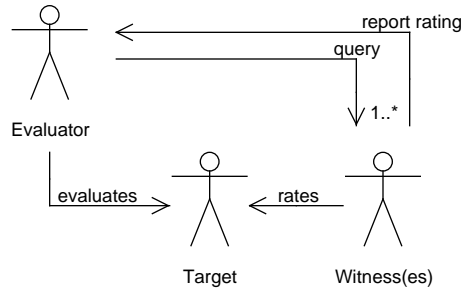


Figure 4. Generic trust model: conceptual relationships

- Decentralized systems

In each model, the agent evaluating the trustworthiness of another is called an *evaluator*, while the evaluated agent is called the *target*. The evaluator may query *witnesses* with direct experience with the target. The witnesses respond to the evaluator’s queries with ratings. The collective ratings impact the target’s reputation, which the evaluator uses along with internal criteria to determine the target’s trustworthiness. Figure 4 demonstrates a generic view of the relationships between the evaluator, target, and witnesses.

Our research relies on a centralized reputation system with the goal of indicating the level of service one agent expects to get from another. The basis of this approach is the same as that of online reputation mechanisms such as eBay [132] and Amazon. Following an interaction, a witness *conceptually* rates the target according to the perceived level of service received. The rating is stored centrally and combined with other ratings to allow the centralized evaluator to determine the resulting reputation. This reputation can then be used as a criterion by which other agents decide whether to engage the reputation holder, or how to treat any information provided by the reputation holder.

Some explanations are in order: why centralize this component when the reason

for using the multi agent system design paradigm is to leverage the advantages of a distributed approach? What is meant by saying the witness *conceptually* rates the target?

Recall that one of the reasons for using a distributed system of mobile agents for intrusion detection is to maximize packet inspection and minimize bandwidth involved. While the volume of traffic flowing into and out of the network as a whole is too large for more than a sparse sampling by the network-based IDS, each node receives, on average, a much smaller volume of traffic amenable to full inspection by a resident ID agent. The distillation of this traffic, as performed by each agent, into feature value sets to be shared with peers represents the first stage of bandwidth minimization. The sharing of only classification *results* with the central agent controller represents the second. The user receiving the majority result from a single entity (the agent controller) at a single node represents a third stage of bandwidth minimization. The *reputation* system is centralized in the agent controller to eliminate the need for additional messages that would otherwise be exchanged between agents. Observe that the witness *conceptually* rates the target in this reputation scheme: in fact, it is the agent controller that determines the witness' rating by proxy. The agent controller has all of the information necessary for the job. With each agent's combined and local classification results, the agent controller can both calculate the majority classification as well as calculate the ratings for each agent that shared information with peers. The agent controller can then apply these ratings to each agent's reputation, which reputation is then available, at a single location, to all other agents in the system.

In short, the centralized reputation system simplifies the MAS design and is in line with the goal of reducing the bandwidth incurred by the MAS.

Another example of a multi agent system employing reputation is SPORAS [155],

which updates reputations with each receipt of a rating according to the following principles [80]:

1. New users start with a minimum reputation value, building up reputation during their activity on the system
2. The reputation value of a user never falls below the reputation of a new user
3. After each transaction, the reputation values of the involved users are updated according to the feedback provided by other parties, which reflect their trustworthiness in the latest transaction
4. Users with very high reputation experience much smaller rating changes with each update
5. Ratings must be discounted according to age so that the most recent ratings have more weight in the evaluation of a user's reputation

The first two rules discourage users from simply creating new accounts to escape the consequences of a series of bad interactions. But one can imagine environments in which migration should be encouraged, such as when the service an agent can provide is dependent on the agent's location. In such cases, a migration threshold may be set below the restart value. In this way, reputation may be used to govern the mobility patterns in the multi agent system. This is one of the desired behaviors of our system.

2.7 Evolutionary Computation

There are many ways to introduce additional flexibility into the multi agent IDS. The values agents use to rate each other in the reputation system can be changed. The parameters of the agent's classifier can be changed as well as the classification technique itself. But as the system's degrees of freedom increase, finding an optimal

set of parameters quickly becomes intractible. Evolutionary computation addresses this search challenge and provides a stochastic approach to achieving notions of self organization discussed in Section 2.5.

2.7.1 Evolutionary Algorithms: Concepts and Formalism.

The generalized notion of an Evolutionary Algorithm (EA) is applicable to several representatives; chiefly, Genetic Algorithms (GAs), Evolution Strategies (ESs), and Evolutionary Programming (EP) [8, 39, 140]. EAs draw inspiration from organic evolution as a means of searching for competitive solutions in situations where efficient search for the optimal solution is elusive (e.g. NP-hard problems, such as feature subset selection - see Section 2.2.1). They are one of a wide variety of algorithms inspired by biological processes, which include ant colony optimization ([3]), artificial immune systems ([38]), and many others.

Genetic algorithms became popular through the work of Holland since the 1970s [77] and emphasize *recombination* (producing new candidate solution vectors from pieces of existing solution vectors) as a search strategy. Evolution strategies, on the other hand, emphasize *mutation* (modifying one or more elements of an existing solution vector) and rarely employ recombination [140]. For a survey of evolution strategies and how they differ from genetic algorithms, see [9].

In evolutionary computation, the process involves initializing a *population* of candidate solutions, where each solution is a vector of parameters proposed for the system whose performance is being optimized. Each member of the population is evaluated by supplying the parameters to the system and measuring performance to determine the member's *fitness*. The measure of performance may be a single value (which could either represent a single objective or a weighted sum of objectives) or a vector (e.g. in the case of multi-objective optimization). Fitness values are used as *selection*

criteria to determine which members of the population should survive into the next generation. Selected members may then undergo *recombination* and/or *mutation* to produce new candidate solutions. Recombination produces offspring by combining the parameter values of “parent” solutions in some way. Mutation only involves one “parent” and simply changes specific parameter values as a means of exploring the solution space. The resulting population may be a mix of old and new solutions. All are evaluated as before, after which selection happens again, and so on for potentially many generations until some criteria is satisfied (e.g. some set number of generations have completed, some performance criteria has been met, or performance ceases to improve).

Bäck presents a useful EA formalism [8]. Optimization as a minimization of a function $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $M \neq \emptyset$ consists of searching for $\vec{x}^* \in M$ such that $f(\vec{x}^*) > -\infty$ and

$$\forall \vec{x} \in M : f(\vec{x}^*) \leq f(\vec{x})$$

This is easily converted when optimization requires a maximization. Regardless, the goal, usually unrealizable within time and other resource constraints, is to find the global optimum \vec{x}^* for the objective function f within the feasible region M .

The formal definition of a generic, single objective evolutionary algorithm is provided in Appendix 1.1.

For multiple criteria decision making (MCDM) problems, f is of the more general form $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^k$, where again $M \neq \emptyset$ but $k > 1$ [39, 96, 97]. The implication is that in most cases, there is no single solution $\vec{x}^* \in M$ which produces the global optimum for every criterion of f . Instead, there exists a set of non-dominated solutions known as the *Pareto* set, such that the quality of a solution can be improved with respect to a single criterion only by becoming worse with respect to at least

one other criterion. An algorithm that employs evolutionary computation to tackle MCDMs is known as a Multi Objective Evolutionary Algorithm (MOEA).

Two examples of popular MOEAs are the Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) and the Strength Pareto Evolutionary Algorithm 2 (SPEA2). They differ in the way members of the population, each representing a solution, are ranked after fitness evaluation. The challenge is that for a multiple criteria decision making problem, there are several dimensions of fitness. How should one combine the fitness values across all dimensions to produce a rank ordering of the population? NSGAII and SPEA2 each answer in a different way.

From [39], NSGA-II first groups individuals according to nondomination strata. Individuals in the same group share the same fitness value, which is proportional to the population size. Individuals in the “lowest” stratum (when each objective is a minimization objective) receive the lowest rank, and the highest fitness value. The selection operator is elitist, taking individuals from the lowest rank possible until the required number are selected. One additional factor is considered: the crowding distance. This is the distance, in the objective space, between a candidate for selection and other individuals already selected. By factoring this in the selection process, NSGA-II ensures diversity of solutions and explores the fitness landscape.

SPEA2 maintains an archive of “best” individuals seen so far in terms of fitness. It assigns fitness to each individual on the bases of 1) how many individuals in the archive and the current population dominate it; and 2) how many other individuals in these sets it dominates.

Both NSGA-II and SPEA2 are proposed for examination for the purpose of finding the best way to rate agents in the reputation system described in sections 2.6 and 3.5.5.

Coello et al. [39] provide an overview of the use of evolutionary algorithms for

Table 1. Characteristics of multi-objective metaheuristics frameworks (from [103])

Framework	Problems		Statistical Tools		Parallel	Type	Language	License
	Cont.	Comb.	Off-line	On-line				
jMetal	yes	yes	yes	no	no	open	java	free
MOEA for MATLAB	yes	no	no	no	yes	closed	MATLAB	comm.
MOMHLib++	yes	yes	no	no	no	open	c++	free
PISA	yes	yes	yes	no	no	closed	any	free
Shark	yes	no	no	no	no	open	c++	free
ParadisEO	yes	yes	yes	yes	yes	open	c++	free

multi-objective problems. Their work also examines a recurring theme in multi-objective evolutionary algorithm research: performance comparison on a variety of benchmark problems with different characteristics. For examples of various applications of evolutionary algorithms, see Appendix 1.1.1.

2.7.2 Evolutionary Algorithms: Software.

Popular software frameworks for multi objective metaheuristics research include [103]:

- jMetal [47]
- MATLAB MOEA toolbox [141]
- PISA [25]
- MOMHLib++ [83]
- Shark [81]
- ParadisEO-MOEO [103]

Characteristics of these frameworks are presented in Table 1, from [103]. Distinguishing characteristics include: the type of problems they can handle (continuous, combinatorial); availability of statistical tools (including performance metrics); facilitation of parallel algorithms; framework type (closed or open); programming language; and the license type (free, commercial).

With the exception of MATLAB’s MOEA toolbox, all of these frameworks are freely available. They greatly facilitate experimentation by providing implementations of many of the MOEA algorithms found in the literature. The open frameworks allow the user to modify existing implementations or reuse existing components when designing new algorithms.

Thus, the use of an existing framework clears many implementation hurdles standing in the way of leveraging evolutionary algorithms in pursuit of a self-adaptive, or ‘self-organized’ system.

2.8 Self Organization and Emergence

The preceding two sections discussed reputation and evolutionary computation as techniques for achieving a measure of self-organization. It is important to distinguish *self-organization* from *emergence* because both concepts are independently beneficial but often erroneously conflated. In recent years, as the notion of self organization has been refined, several authors have argued for differentiating it from emergence [41, 71, 128].

As De Wolf and Holvoet explain [41], emergence and self-organization each emphasize very different characteristics of a system’s behavior. They may exist in isolation or together in a dynamic system.

Emergence is defined loosely as the phenomenon where global behavior arises from the interactions between local parts of the system. The proposed definition by De Wolf and Holvoet:

A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel with respect to the individual parts of the system.

The ‘emergent’ referred to in this definition denotes the properties, behaviors, structures, patterns, etc. that result from the process of emergence. The macro-level is the view of the system as a whole, while the micro-level considers the components of the system.

The required “novelty” of the emergents declares that the emergents have no explicit representation at the micro-level. They are not reducible, in terms of explanation of the phenomena, to the micro-level parts of the system—we cannot reduce the whole to a sum of its parts (reductionism) [41, 76].

The working definition of self-organization proposed by De Wolf and Holvoet:

Self-organization is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control.

Systems that exhibit self-organization demonstrate an increase in order with autonomy. They are robust in the sense that they adapt to handle changing conditions. Finally, they are dynamic, which is to say, far-from equilibrium.

Though these concepts are clearly distinct and can exist independently of each other, they are often combined, particularly in the case of multi agent systems. For example, a self-organization mechanism may autonomously put the agents in the configuration that produces a desired emergent. This is precisely the situation for the multi agent system presented in subsequent chapters: via reputation, the agents exhibit self-organization in the activity of distributing themselves across the network. The resulting distribution is the emergent.

2.9 Evaluation of Classification and Heuristic Systems

The concepts discussed in this chapter contribute to the development of a system that is primarily a classifier. It includes a heuristic in form of a reputation system as

it indirectly searches for spatial agent distributions resulting in optimal classification performance. Attention in this section turns to the evaluation techniques appropriate for such a system.

Computational experiments with algorithms, note Barr et al. [16], are usually undertaken 1) to compare the performance of different algorithms for the same class of problems; or 2) to characterize or describe an algorithm's performance in isolation.

Comparisons with state of the art algorithms are recommended for demonstrating whether and where new algorithms chart new territory in terms of performance. In many cases, however, other methods do not exist. Nevertheless, in such cases the researcher can demonstrate a heuristic's effectiveness as compared with a more primitive or more general method serving as a baseline [16].

In experiments that characterize rather than compare a given algorithm, individual factors can be analyzed for impact on the algorithm's performance. Experimentation techniques applied to such factor analysis include *factorial design* and *analysis of variance* [16].

A complete factorial experiment includes all possible factor-level combinations in the experimental design [111]. The order in which each factor-level combination is tested (or the testing units to which the combination is assigned) is typically randomized. As Mason et al. explain, randomization affords protection from bias by tending to average the bias effects over all levels of the factors in the experiment [111].

Of interest is determining whether two different factor-level combinations produce statistically significant differences in the populations of values obtained for the response variables. One can set up a hypothesis test checking for equality in selected test statistics (such as the mean or the median) and obtain a probability of the truth of the hypothesis given observed values in each population.

Statistical tests for accomplishing this task include the T test and the Wilcoxon

rank-sum test [121]. The T test is the most powerful under the assumptions of normally distributed random variables with equal but unknown population variances. These restrictive assumptions are not always reasonably acceptable. In such cases, the nonparametric Wilcoxon rank-sum test is a popular alternative [121].

For pair-wise comparisons of various factor-levels, these tests are simple to implement and sufficient for analysis of a proof-of-concept system. A more general *factorial design* approach examines the strength of individual and joint factor effects in all factors involved [123, 121, 111].

In Section 5.2.1, the Wilcoxon rank-sum test is used to establish the probabilities of population median equality between various factor-level combinations (for example, three agents using reputation with decay vs. three agents not using reputation). Future research should explore joint factor effects as the designed flow-based multi agent IDS progresses beyond the proof-of-concept phase toward real-world operational capability.

2.10 Summary

This chapter considers, in Section 2.1, the AS-level Internet topology and traffic modeling requirements in order to conduct the desired relevant research. This is followed in Section 2.2 by a discussion of the prototypical Pattern Recognition system as a template for what our system must implement and accomplish. In particular, our research implements a classification system. Evaluation of classification systems is consequently discussed in Section 2.9. Section 2.3 examines pattern recognition in the context of identifying malicious network activity, which is known as intrusion detection; particular consideration is given to flow-based techniques. In Section 2.4, a brief discussion of multi agent systems and their applicability to intrusion detection is presented. Section 2.5 presents a list of system intervention points providing design

options when building a multi agent IDS. To achieve a degree of ‘self-organization’ as defined in this list, Section 2.6 considers the notion of reputation. A far broader way to achieve self-organization via the use of Evolutionary Computation is presented in Section 2.7. Also, the evolving definition of self-organization and its relationship to and distinction from emergence is addressed in Section 2.8.

The next two chapters employ these concepts in the presentation of multi agent system designs for detecting and classifying network attacks.

III. MFIRE: Network Simulation and Multi Agent System Design

A new system design is required for autonomous classification of network attacks in a live, albeit simulated, network. We observe that the difficulties of ID system design is an ongoing process of enlightenment due to the approximation of the network environment and the reaction to it. Thus, the multi agent system paradigm, with several performance-enhancing details, is leveraged in this second design iteration in order to maximize the performance. Differences between this and the first design iteration are presented in Chapter IV. The agents are designed to be mobile and cooperative in terms of sharing feature observations. Over a series of simulated attacks, the integrated system searches for a ‘good’ distribution of agents via a ‘reputation’ system. The parameters of this reputation system are improved a priori via a multi-objective evolutionary algorithm.

In this chapter, the high-level methodology and detailed software design of our second iteration network simulation and the MAS-based network activity classifier (MFIRE) are presented. Section 3.1 provides the overall design to allow details in proceeding sections to be placed in context. Section 3.2 discusses the formal classification of our solution and the corresponding parameter search space. Section 3.3 justifies the selection of the MASON Discrete Event Simulation (DES) engine as a framework for the network simulation environment presented in Section 3.4. Section 3.5 elucidates the design of the multi agent system by describing its components and facilities for self-organization. Additional details are provided in Appendix B.

3.1 Network Simulation and Multi Agent System Design Overview

The design of a network simulation environment suitable for our purposes involves the representation of essential network components and operations. Specifically, nodes must route traffic, generated by processes, over links with limited capacity, in a topology reflective of what is seen in the real Internet (see Section 2.1.2). Some of the processes represented are ‘normal,’ generating traffic according to distributions seen on the real Internet (see Section 2.1.3), while other processes represented are ‘malicious,’ causing congestion on network links, systematically extracting information regarding potential vulnerabilities of network nodes, and/or spreading copies of themselves to other nodes without authorization.

To enable the properties described in such a simulated network environment requires a representation of traffic as content-bearing packets, facilities for delivering these packets to specific destination processes, and facilities for instantiating a network complete with its nodes, links, processes, and properties of each (e.g., respectively routing tables, link capacities, and traffic-generation and response behaviors). To aid understanding at a high-level of traffic patterns present from moment to moment during a simulation, visualization facilities should also be considered.

This section presents the package hierarchy providing a framework in which to place the required representations of these concepts. In addition to the network simulation environment, a multi agent classification system is designed as a set of processes, with components including agents and an agent controller. To support the agents’ classification responsibilities, interfaces are designed for classification techniques and feature definitions, enabling changes in detailed implementations without requiring changes to the system architecture.

Figure 5 presents a general view of the package hierarchy involved in the simulation.

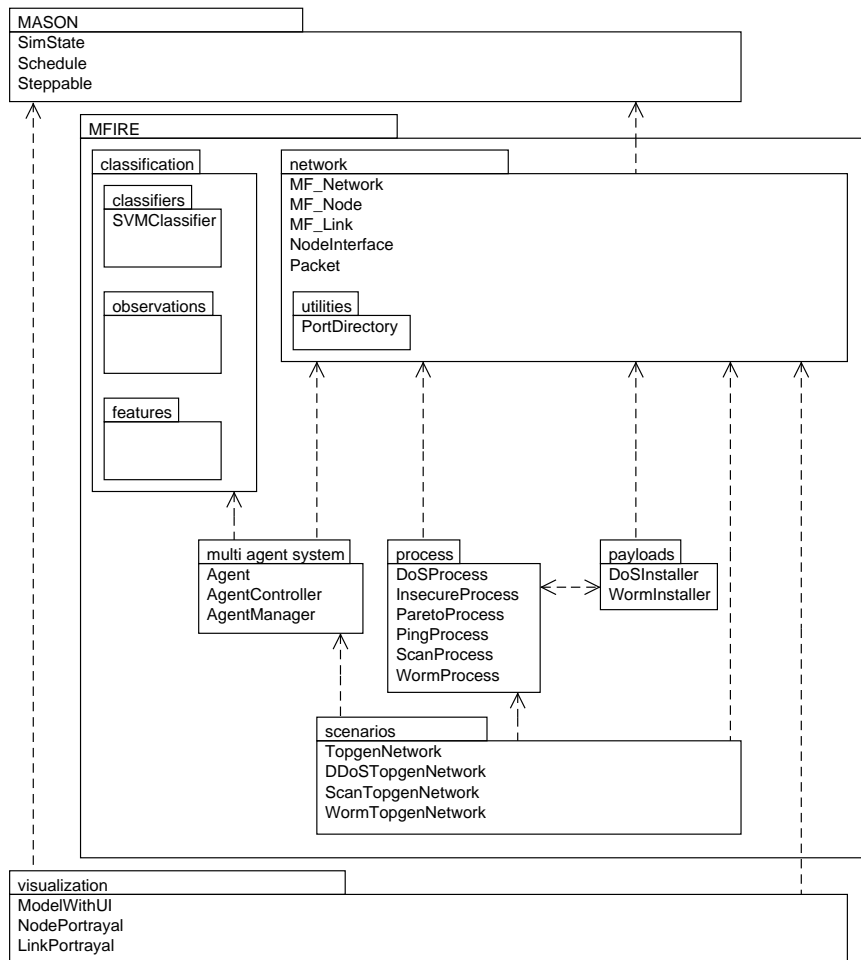


Figure 5. MFIRE package diagram.

A few of the more notable classes and interfaces are shown as members of their enclosing packages. The controlled, one-way dependencies between the visualization layer, the *domain layer* (labeled ‘MFIRE’), and the *application layer* (‘MASON’) exhibit a software engineering principle known as *Model-View Separation* [102]. This principle states that domain objects should not have direct knowledge of view (UI) objects. It allows the visualization layer to be changed without requiring any changes in the domain or application layer.

As shown in Figure 5, the domain layer consists of the following groups of classes:

- Network - includes representations of physical domain entities of interest. This is the ‘core’ of the simulation.
- Scenarios - concrete realizations of the abstract MFNetwork. The prominent class is the TopgenNetwork, which includes facilities for loading a network produced by the Topgen AS-level Internet topology generator. Each class in this package is characterized by a unique set of Processes initially running on a subset of the nodes.
- Processes - These are analogous to the networked applications on the real Internet. Each Process runs on a host node and may receive and/or generate traffic.
- Payloads - Specially crafted payloads execute code when opened by a certain receiving processes. These payloads can be written for legitimate purposes, such as Remote Procedure Calls (RPC), but our focus is on payloads that install malicious processes on the receiving node.
- Multi agent system - This package includes the “worker bees” - the Agents, the “queen bee” - the AgentController, as well as AgentManagers with special local oversight of any Agents on the same host node.

- Classification - Agents make use of entities in this package to make local classification decisions. Included are the classification algorithms, enclosed in the ‘classifiers’ package, and the observations and features used. Strictly speaking, both observations and features are statistics-based calculations, but we distinguish the observations as being more “raw” than the features. By ‘feature’ we imply there is something composite in its nature - it may be an average of observation values or the result of some other series of mathematical operations on the observations and/or other features.

At the top of Figure 5 is the MASON discrete event simulation engine package, which provides many vital facilities for the execution of the simulation as well as the visualization of the same. The details of the visualization are specified via entities in the visualization package at the bottom of the diagram.

Figure 6 provides a class diagram for some architectural detail of the more prominent aspects of the domain representation. This is not intended to provide a comprehensive listing of the classes nor the attributes and methods of each class. Rather, expressed are some of the essential class associations and hierarchies that drive the network simulation.

3.2 Formal Problem Specification

In any implementation of the high-level design, ultimately we seek to 1) maximize the classification accuracy of all attack classes; and 2) use minimal network bandwidth. This requires an effective spatial distribution of a limited number of agents, where each agent uses a limited number of features to make local classification determinations. These determinations are then shared with a centralized controller. This section defines the problem formally to provide an unambiguous template for design implementation.

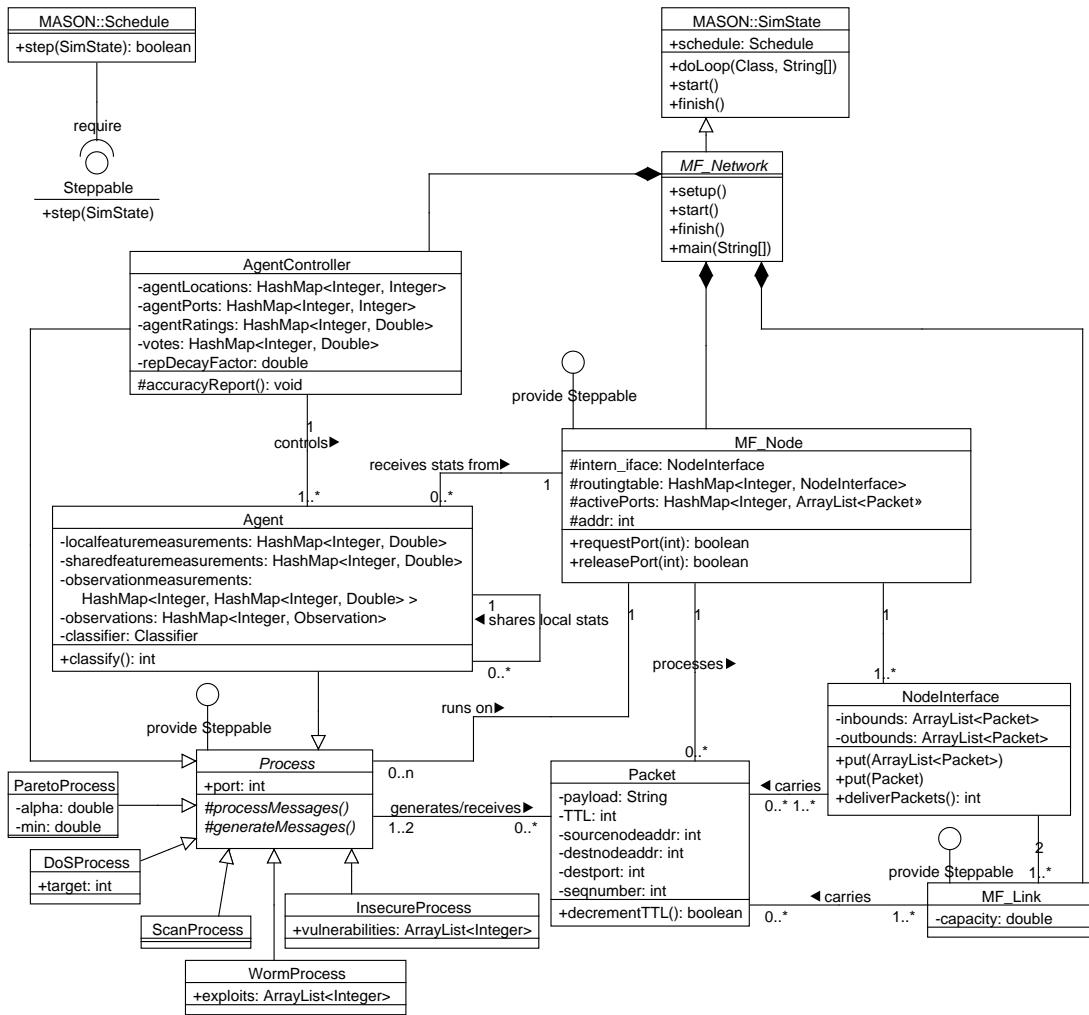


Figure 6. MFIRE class diagram.

We denote the set of network activity classes as \mathcal{A} . An arbitrary set of p features arising from any given instance of \mathcal{A} is denoted as the vector X . $A(X)$ yields the label of the class that produced X . One does not have $A(X)$ for arbitrary X , but seeks an accurate model, denoted $\hat{A}(X)$. Maximizing the classification accuracy motivates selection of $\hat{A}(X)$ and associated parameters θ . One of these parameters is the feature subset selection vector $S_X \in \mathbb{B}^p$, where $S_{X_j} = 1$ if the corresponding component feature X_j is used, $1 \leq j \leq p$. Note that searching for the optimal feature subset among 2^p possibilities is known to be NP-Hard [5]. Other parameters involve the number and distribution of agents in the multi agent classifier.

Intuitively, there is some competition between the multiple objectives of minimizing bandwidth and maximizing classification accuracy. One or the other is adversely affected by increasing or decreasing the number of agents as well as the amount of information exchanged between agents.

If $d(a)$ generically denotes the duration of scenario a , the multi agent system is composed of z agents, and $m_i(q, r)$ denotes the number of bytes sent from agent q to agent r on iteration i of the scenario, then the bandwidth $B(a)$ is:

$$B(a) \triangleq \sum_{i=1}^{d(a)} \sum_{q=1}^z \sum_{r=1}^z m_i(q, r) \quad (13)$$

If the dominating reason for sending messages between agents is to share feature measurements, then the size of each message is proportional to the number features shared. Thus:

$$m(q, r) \propto \sum_{j=1}^p S_{X_j} \quad (14)$$

From equations 13 and 14 it is clear that minimizing the bandwidth depends on minimizing the number of agents z and / or the number of features used, which is

$\sum_{j=1}^p S_{Xj}$. These parameter values are reflected in θ . Obviously, one should not reduce them such that the Expected Prediction Error, equation 7 (Section 2.2) is injuriously impacted:

$$\text{EPE} \triangleq \text{E}[L(A, \hat{A}(X))] \quad (7)$$

This motivates the search for an effective distribution of a limited number of agents. Certain locations in the network yield feature measurements which are better able to distinguish the ongoing activity class; we expect these locations to be highly-connected nodes.

Simply, there are two major areas of concern: 1) selection of the classifier model; and 2) the mechanism by which agents agents distribute themselves throughout the network.

3.2.1 Classifier Model Selection.

The objective is to select \hat{A} , the classifier model, such that for inputs X and ‘true’ outputs $A(X)$, equation 7 is minimized.

We use the *zero-one* loss function (where an equal penalty is applied to all misclassifications). This leads to the use of equation 9 (Section 2.2), which guides specification of $\hat{A}(X)$ as selecting the most probable class $a \in \mathcal{A}$ given the input:

$$\hat{A}(X) = \mathcal{A}_k \text{ if } P(\mathcal{A}_k|X = x) = \max_{a \in \mathcal{A}} P(a|X = x) \quad (9)$$

The challenge lies in the fact that $P(a|X = x)$ is unknown. An estimation of this probability involves the selection of the classification algorithm, a set of classifier parameters θ appropriate for the selected algorithm, and a training process.

We select the Support Vector Machine technique [142, 144, 143] as a starting

point for $\hat{A}(X)$ due to its high generalization performance (see Section 2.2 and [34]). Fully defining $\hat{A}(X)$ involves selection of θ , which includes the feature subset selection vector S_X .

3.2.2 Spatial Distribution of Agents.

The search for the most effective spatial distribution of the agents is akin to the feature selection problem (see Section 2.2.1). For the graph $G = (V, E)$ corresponding to the network of interest, where V is the set of vertices and E is the set of edges, the objective is to find the subset of V that optimizes equation 7. The associated vertex selection is denoted $S_V \in \mathbb{B}^{|V|}$, subject to the constraint that $\sum_{j=1}^{|V|} S_{V_j} = z$, where z , again, is the number of agents in the multi agent system. Observe that selecting z vertices from the set V entails a search space of

$$\binom{|V|}{z} = \frac{|V|!}{z!(|V| - z)!} \quad (15)$$

possibilities.

Section 3.5 discloses the mechanism for motivating agents to seek good ‘vantage points.’ It relies on a ratings and reputation system for the agents. Agents that consistently share heuristically ‘good’ information accrue high reputation and achieve stability, while agents that do not suffer a loss of reputation until prompted to migrate to a neighboring node.

This discussion of formal problem specification is preceded by the high-level network simulation and MAS design. The proceeding sections present intermediate and low-level design and implementation. Although such details are useful in recognizing the design intricacies, one can, still without them, appreciate the discussions of the remaining chapters.

3.3 Discrete Event Simulation Engine: MASON

There is a wide variety of discrete event simulation engines available for building a simulation (see Section 2.1.4.1). Typically, each emphasizes a certain domain or technique. For example, OMNeT++ [145] is geared toward network simulation, as is ns2 [112] and cnet [114, 113], while Parsec’s emphasis is parallelism [11]. Finally, MASON [106] caters to the needs of multi agent systems simulation.

There are three principle reasons for the selection of MASON as the underlying DES engine:

- This research concerns a multi agent system - MASON’s structural expertise
- MASON does not impose nor even provide a predefined abstraction of real-world computer networks. Our implemented network simulation is customized to focus on prototyping a system able to function in a moderately complex network environment. Several of the discrete event simulation engines mentioned in this section are heavily invested in accurate simulation of real-world protocols, network devices, and applications, but the much higher complexity of these environments introduces networking issues not directly relevant to MFIRE’s initial implementation.
- We have some prior experience with MASON. This research began as a way to complement the attack mitigation capabilities of Holloway’s Self Organized Multi Agent Swarms (SOMAS), [78]. SOMAS operates in a MASON-based network simulation.

Observe that although MASON is designed to handle large numbers of agents in complex environments [106], it is not explicitly an agent framework, such as the frameworks provided by JACK, Cougaar, JADE, and others [105]. These frameworks

in some cases (e.g. JACK [79] and JADE [20]) provide compliance with the interoperability standard called the Foundation for Intelligent Physical Agents (FIPA) [21]. Such agent frameworks should be explored for use by MFIRE in future research.

3.4 Simulated Network Design - MFIRE's Domain of Operation

With MASON having been selected as the network simulation driver, this section discusses the design of the simulated autonomous system network, including its components and topology. Discussion of network traffic is reserved for Sections 3.4.4 and 3.4.5.

3.4.1 Network Simulation Design Objectives.

Design objectives for the physical aspect of the AS-level Internet simulation include:

1. Employ a topology representative of the AS level of the Internet
2. Allow for multiple scales of AS networks in terms of maximum link distance
3. Provide visualization facilities to enhance intuitive understanding of simulation execution

The first objective is present because of the impact of topology on system performance, as discussed in Section 2.1.2 and [154].

The second objective introduces flexibility sufficient to accommodate the needs of different potential MFIRE clients. One client could be a federation of universities, government agencies, and Internet service providers within a state. Another client could be the U.S. military. The maximum link lengths involved in connecting the state-wide federation of autonomous systems are much shorter than the links connecting the autonomous systems of a global military.

The third objective is useful for qualitative system and scenario validation. This validation has a boolean output based on whether desired behaviors are observed. For example, with each timestep in our simulation, our visualization paints links some color between green and red (or white and black) depending on the ratio of current traffic to the link’s capacity. Under a Distributed Denial of Service scenario, one should expect to see numerous red (black) links along various paths to the target.

3.4.2 Network Simulation Low Level Components.

The physical network components simulated in this research investigation include:

- Nodes - each node represents an Autonomous System (AS). Internal to an AS is a collection of routers, switches, firewalls, and edge devices, including servers and clients. These devices are all abstracted into one node in our simulation, represented by the `MF_Node` class in Figure 6. Nodes route traffic via routing tables, initialized via the Floyd-Warshall shortest path algorithm [53]. This is analagous to gateway routers employing BGP (see Section 2.1) on the real Internet, though with BGP, policy decisions often trump routing efficiency (competing Internet service providers, for example, may refuse to allow ‘through’ traffic without compensation). Each node is addressable by a unique identification number. Nodes provide resident processes with basic communications facilities, such as the `send()` method, which creates and sends packets. Nodes implement the `Steppable` interface and therefore supply a `step()` method invoked on each timestep of the simulation. This method primarily switches packets from the inbound queues of all `NodeInterfaces` to the outbound queues of `NodeInterfaces` identified in the routing table, via lookup on the packet’s destination address.
- `NodeInterfaces` - These are intermediaries between Nodes and 1) Links; or 2) Node’s resident Processes. The first case includes all external-facing interfaces,

while the second describes the Node's internal interface. Each is an entry/exit point. All NodeInterfaces have an inbound queue and an outbound queue. The inbound queue is read by the attached Node and written to by the attached link. The outbound queue is read by the attached link and written to by the attached Node.

- Ports - associated with nodes, ports are the communication end points for processes running on servers and clients. In the real world, each computer typically has many thousands of ports associated with each transport-layer protocol. For example, there are 2^{16} ports available for Transmission Control Protocol (TCP) and another 2^{16} for User Datagram Protocol (UDP), the number being fixed by the width of the port field in the segment, respectively datagram header [126, 127]. In our simulation, each port on an AS node corresponds with a port on an arbitrary host internal to the AS.
- Port Directory - Certain “well-known” ports are reserved for special purposes. This is the case with the real Internet, for which a list is maintained by the Internet Assigned Numbers Authority (IANA) [2] specifies how certain ports are to be used, such as port 80 for Hyper Text Transfer Protocol (HTTP) traffic. When these standards are adhered to, finding public services is greatly simplified. Also, filtering of certain expected types of traffic becomes simple. Observe that, in our simulation, some ports are reserved for components of the multi agent system.
- Links - links in our network simulation are strictly point-to-point and connect autonomous systems together. Links are full duplex but have finite bandwidth. Depending on the scale of the simulation, links may vary in length, affecting propagation delay. One of three scales is specified at the start of each simulation:

- LOCAL - All links have the same unit length. Packets traverse these links in one step of simulation time.
- REGIONAL - Link lengths vary from one to ten units. This is useful when the simulated AS topology spans a continent.
- GLOBAL - Link lengths vary from one to 100 units. This is appropriate for simulation of an AS topology in which some of the nodes are satellites in geostationary orbits, for which propagation delays can indeed be on the order of 100 times those of terrestrial links.

Scale is realized with each link being composed of sublinks. Links implement the `Steppable` interface. Each timestep, when the Link's `step()` method is called by the `Schedule`, the Link causes each Sublink to pass its traffic to its adjacent Sublink (or, ultimately, `NodeInterface`).

- Processes - these include processes that strictly generate traffic for the benefit of the simulation as well as classifying agents that generate actual communication traffic (primarily to share observations). All processes run on nodes and must be assigned a port before they can send and receive packets. Processes implement the `Steppable` interface. When `step()` is called, the Process first receives and processes traffic, and then generates outbound traffic.
- Packets - Each packet consists of the following:
 - Source node address - identifies the Node of origin
 - Source port - the port used by the sending Process
 - Destination node address - identifies the Node hosting the intended recipient Process
 - Destination port - communication endpoint for the intended recipient `Process`

- Sequence number - Facilitates sending messages spanning multiple packets
- TTL - Time To Live - the number of hops allowed before some intermediate Node discards the packet. This mitigates problems arising from routing loops induced by congestion or misconfiguration of the routing tables.
- Payload - a string containing the message the sending Process wishes to pass to the intended recipient. The format of this message is entirely up to the communicating processes.
- size - Indicates the size of the payload, in numbers of characters, if a real payload is used. If a real payload is not required (e.g. to simulate background traffic or junk traffic sent by denial-of-service processes), the sending Process can simply specify the desired size of the packet to be sent, leaving the payload string null and preserving memory.

With the previous component discussion completed, the flow of our simulation can be explained. During initialization, after all network components have been instantiated, all Processes, Nodes, and Links are scheduled to execute associated tasks on every timestep (e.g. generate traffic, process traffic, move traffic). They are prioritized as follows:

- First, Processes handle received traffic and generate new traffic
- Second, Nodes handle traffic by switching packets from inbound queues to appropriate outbound queues or ports
- Third, Links move traffic along component Sublinks toward the NodeInterfaces on either end

The corresponding real-world components, of course, behave asynchronously. The synchronous aspect of the simulation is a design simplification. As stated in [69],

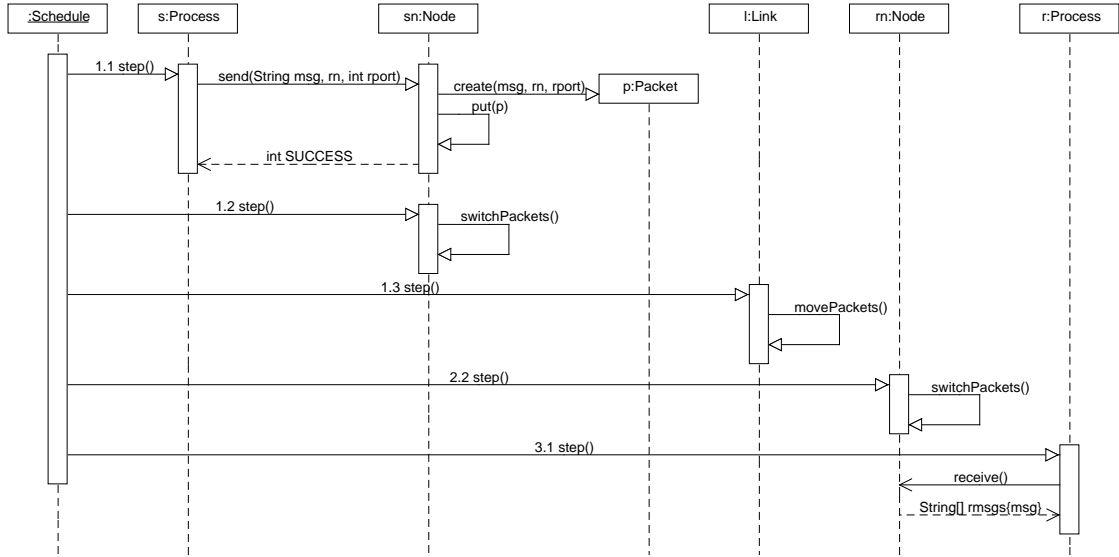


Figure 7. MFIRE sequence diagram illustrating the transmission of a packet

synchronous model composition helps provide structure without introducing non-determinism. A deterministic model relying on pseudorandom number generators allows reproducible tests. All that is required to reproduce a specific test is to preserve the seed with which the pseudorandom number generator is initialized.

It may nevertheless be disconcerting to apply a synchronous model to intrinsically asynchronous situations. It is well-known since Milner’s papers on the subject [120, 119], however, that a synchronous formalism can be used to express asynchrony. Explicit non-determinism and sporadic process activation help bridge the gap between the synchronous model and the asynchronous reality [69]. For examples, see [17, 18]. Attention to more accurate modeling of underlying asynchronous communication processes is reserved for future research.

In Figure 7 the sequence of events involved in packet transmission between Processes on adjacent Nodes is depicted. For clarity, some classes such as the NodeInterfaces and the Sublinks have been left out. Some of the method parameters are likewise omitted for ease of understanding the process. In this Figure, steps are la-

beled by timestep and by the priority of the **Steppable** involved. The sequence of events is as follows:

1. The **Schedule** invokes the sending Process.
2. This Process generates a message and calls the **send()** facility provided by the host Node.
3. The Node assembles a Packet with the message as the Packet's payload. The Node then places the Packet in the inbound queue of its own internal **NodeInterface**. Doing this normalizes the way packets are handled, whether generated internally or received from another Node.
4. The **Schedule** invokes the Node.
5. The Node switches the Packets from the inbound queues of all **NodeInterfaces** (including the internal **NodeInterface**) to the outbound queues of the appropriate **NodeInterfaces** via the routing table.
6. The **Schedule** invokes the Link attached to the **NodeInterface** that is now holding the Packet in its outbound queue.
7. The Link takes the set of Packets from the originating **NodeInterface**'s outbound queue and moves it to the next Sublink. In the case that the scale is **LOCAL**, there is only one Sublink for each side of the full duplex Link.
8. Not shown in Figure 7, the Packet waits with all other Packets on the Sublink until the next time the Link is invoked by the **Schedule**. At this time, assuming the scale is **LOCAL**, the Packets are placed in the inbound queue of the destination **NodeInterface**. If the scale is not **LOCAL** and the length of the Link is greater than one unit, Packets are moved to the next Sublink in the series of Sublinks composing the Link.

9. The `Schedule` invokes the destination Node.
10. This Node switches the Packets as did the originating Node. This time, however, there are Packets that have “arrived.” Each Packet whose destination address matches this Node’s address is placed in a buffer indexed via the port number.
11. The `Schedule` invokes the intended recipient Process.
12. The Process uses the host Node’s `receive()` method to access the Packets in its port. This completes the delivery of the Packet for this example.

3.4.3 Network Simulation Topology.

The question remains how to properly set up a representative collection of nodes and links. Our architecture is extensible in that it permits different ways of initializing the simulation’s network topology, including completely manually. Because the manual approach is unwieldy for all but the smallest networks, automated topology generation is necessary.

From the discussion in Section 2.1.2, we desire a generator that, at a minimum, produces topologies according to “Pareto-type principles,” in which a small number of nodes have many neighbors, while most nodes have a small number of neighbors. While most tools adhere to this, we select TopGen for having incorporated some of the most up-to-date observations about the real Internet AS topology. In particular, the Controlled Distance model is used (see Section 2.1.2 and [138]) to generate 100-Node networks.

Among the tools listed in Section 2.1.2, RealNet is another promising candidate with a different approach. It queries various data sources such as publicly available BGP tables and traceroute records, producing a model using certain current characteristics. Though unavailable for evaluation for our current research, future research

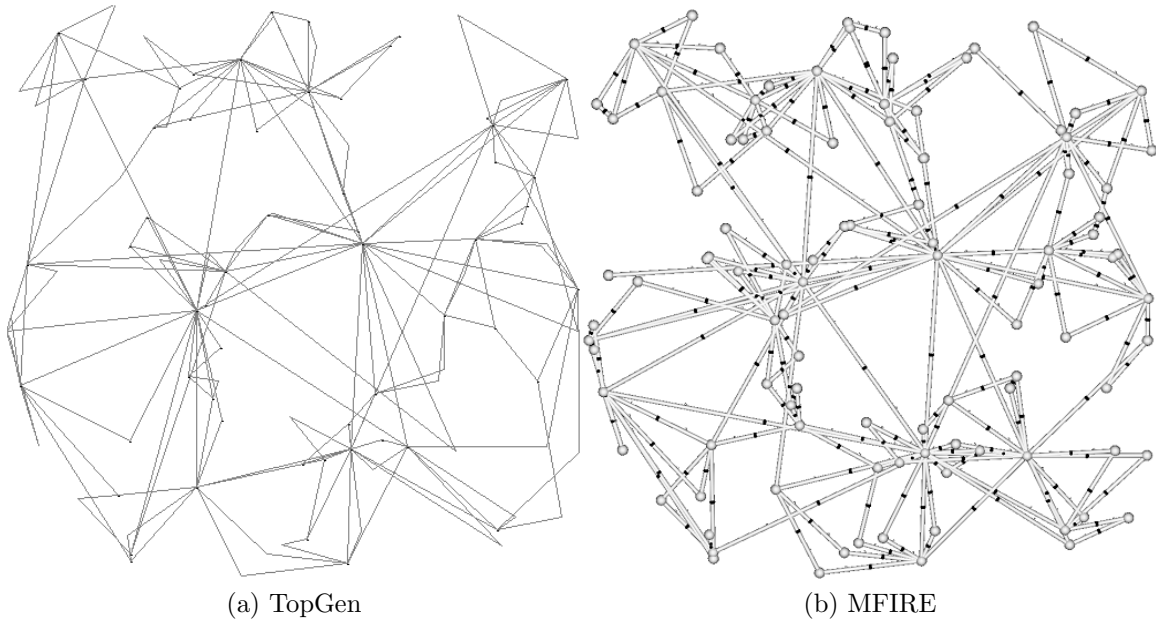


Figure 8. Network produced by TopGen and reproduced in MFIRE

should evaluate RealNet’s applicability as a topology generator for our network simulation.

Figure 8 shows the topology generated and rendered by TopGen as well as the resulting visualization after loading it into MFIRE. Using the Node coordinates supplied in the TopGen file, MFIRE sets link lengths by calculating the Euclidean distance and scaling to the specified range as described in Section 3.4.2.

3.4.4 Network Simulation Traffic Routing.

Once the topology is in place, the Floyd Warshall routing algorithm [53] initializes each Node’s routing table. The result is that when there is no contention for bandwidth, packets always take the shortest path to the destination.

When the preferred link is saturated with traffic, an alternative link is selected at random. If the alternative link is also saturated, the packet is discarded. If nodes were to rerun the routing algorithm when such congestion arises, packet delivery could be made more reliable. This is also a possibility for future research; but, at present, the

random alternative link selection method does allow many more packets to continue to their destination than if there were no route flexibility at all.

3.4.5 Network Simulation Traffic Design.

In this section, the four main traffic patterns are characterized that dominate the simulation. Observe that agents also generate traffic patterns of their own when communicating, but this is normally insignificant in comparison with the patterns described in the next four sections.

3.4.5.1 Simulated Normal Traffic.

Normal traffic follows a Pareto distribution (see Section 2.1.3,[152]). Equation 6 generates a Pareto distributed sequence of values:

$$T = \frac{b}{U^{\frac{1}{\alpha}}}$$

where b is the minimum value. This equation is used by the ParetoProcess class to determine the size of the packet to send to every possible destination on the network. To produce the desired minimum quantity of 0 and the heavy tail observed in real-world Internet traffic [152], we use

$$T = \frac{1}{U^{\frac{1}{\alpha}}} - 1$$

with $\alpha = 2.0$.

When the result is zero, no packet is sent. Otherwise, the result is used to specify the size of a packet with no actual payload. This packet is sent to the destination Node and to a port uniformly randomly drawn from a range of “well-known” port numbers (see Section 3.4.2).

In typical simulations, every Node on the network has a resident ParetoProcess. As a self-similar distribution, one of the properties of the Pareto distribution is that the aggregation of values each drawn from this distribution is also Pareto distributed [152]. This is to say that at any given Node on the network, the cumulative volume of traffic, over all inbound links, exhibits scale-invariant “burstiness.”

The following sections describe malicious traffic patterns that, in isolation, do not demonstrate the same qualities as Pareto-distributed traffic. The hope, elaborated in Section 3.5.4, is that the presence of malicious traffic amidst “legitimate” traffic makes itself known by altering the characteristics of the *aggregate inbound traffic*. When this is the case, there is cause for optimism that a classification algorithm may be able to detect the disturbance.

3.4.5.2 Simulated DDoS Traffic Scenarios.

Flooding-based Distributed Denial of Service (DDoS) attacks rely on overwhelming the target with a flood of traffic [33], as opposed to a semantic-based Denial of Service attack which exploits a system vulnerability [122]. Multiple attack sources send a steady barrage of small packets to a single target node. Even if the target never crashes, it may still be effectively rendered impotent if all communication channels are clogged, shutting out legitimate traffic.

In our current network simulation environment, a DDoS scenario is instantiated with a list of source nodes and a target. On each source, a DoSProcess is installed. On each timestep of the simulation, each DoSProcess attempts to at least saturate the path between its host and the target. The DoSProcess does this by sending 1000 packets, each one of size one-thousandth the capacity of the attached link. This strategy is more effective than attempting to send large packets, which have a higher probability of not being placed on the link at all (when the link is already carrying a

moderately-heavy or worse load).

3.4.5.3 Simulated Scan Traffic.

The objective of scanning is to provide a potential attacker information about a target network and / or specific hosts on the network. Scans may be described as *horizontal*, in which single ports are tested on a wide variety of hosts, or *vertical*, in which many ports are tested on a single host, or *block*, which is a combination of horizontal and vertical [139].

The ScanProcess in MFIRE simulates a UDP scan [146]. The choice of protocol involved in the scan has certain implications. Unlike TCP, UDP is a connectionless transport layer protocol. There is no “handshake” involved with setting up a connection (e.g. the SYN, SYN/ACK, and ACK packets involved in setting up a generic TCP connection). Packets sent to a process listening on a UDP port are simply handled - or not! When a process relies on UDP for communication, if it desires any of the service guarantees attempted by TCP, it must implement them itself in the application layer (the payload of the UDP datagram encapsulated in an IP packet).

Consequently, a UDP scan results in some ambiguity when the scan packet is sent to a given port at a particular address and no response comes back. Ordinarily, if the port is closed, the host may send back an ICMP “port unreachable” message. The UDP scanner can confidently mark that port as “closed.” If the scanning packet conforms to something expected by a legitimate UDP application (say, a DNS query for port 53), and a response is received, the port is marked “open.” When no response is received after a certain time, it may be that the response was blocked by a firewall, or that the host is configured not to send back ICMP port unreachable messages. The listening Process may have stalled for some reason. Whatever the reason, the UDP scanner often simply marks the port as “open / filtered.” It may later try different

```

[1] scanned nodes [18] - [19], ports 40 - 60

[18] profile:
    40    closed
    41    closed
    42    closed
    43    closed
    44    closed
    45    closed
    46    closed
    47    closed
    48    closed
    49    open
    50    closed
    51    closed
    52    closed
    53    closed
    54    closed
    55    closed
    56    closed
    57    closed
    58    closed
    59    closed
    60    closed

[19] profile:
    40    closed
    41    open/filtered
    42    closed
    43    closed
    44    closed
    45    closed
    46    closed
    47    closed
    48    closed
    49    closed
    50    closed
    51    closed
    52    closed
    53    closed
    54    open
    55    closed
    56    closed
    57    closed
    58    closed
    59    closed
    60    closed

Scan took 290 steps to complete.

```

Figure 9. MFIRE: Example block scan conducted by the ScanProcess class

application probe packets hoping to find other listening services [107].

This is essentially how our ScanProcess operates. It is initialized with a range of addresses and ports (supporting vertical, horizontal, or block scans). A ‘rate’ parameter controls how many packets it sends each timestep. To each $\langle destination, port \rangle$ -tuple, it sends a packet with “CONNECT” in the payload. The destination Node is configured to respond with “UNREACHABLE” when no Process is listening on the port. If a Process is listening, in most cases the MF_Process subclass ignores these packets and sends no response. The one exception is the InsecureProcess. When it receives “CONNECT”, it replies with “OK.”

Figure 9 depicts a block scan involving two nodes and a range of 21 ports. From

the description of how the Process and nodes in the simulation respond to the probe packets, it is clear that an InsecureProcess is listening on port 49 on the Node with the address of 18, while the Node at address 19 is hosting an unknown Process on port 41 and an InsecureProcess on port 54. The scan is conducted from the Node at address 1.

3.4.5.4 Simulated Worm Traffic.

A computer worm is a piece of malware that can replicate itself over a network with no user intervention. For many worms, replication is the only goal. The worm attacks vulnerable systems and installs complete copies of itself on compromised systems. Worms may be characterized by aggressiveness and spreading methodology [150].

Important worms include Code Red, Slammer, Blaster, Sasser, Nimda, and Mydoom. The earliest major Internet worm was Morris, which took down major sections of the Internet in 1988 by compromising vulnerabilities in the Berkeley Standard Distribution (BSD) UNIX and derivative operating systems [54, 48]

The worm implemented in MFIRE is simple and serves as a first test case (more complicated worms are possible and should be investigated in future research). The worm is initialized with:

- A range of target Node addresses
- A list of integers corresponding to simulated vulnerabilities. These are analogous to identifiers in the Common Vulnerabilities and Exposures (CVE) database [110]. Many worms rely on a single vulnerability to which many hosts are thought to be exposed, but more sophisticated worms use several. Stuxnet, for example, uses four [51].
- A list of target ports. Most real-world worms target popular services listening

on specific ports.

- A rate parameter controlling how many attack packets to send each timestep
- A delay parameter controlling how long the simulation runs before the worm starts attacking and spreading

In a typical worm attack scenario, the attack surface is initialized by first setting up several active vulnerabilities in the environment. Next, InsecureProcesses are set up at every Node. Each InsecureProcess is initialized with a random subset of the active vulnerabilities. The InsecureProcesses are furthermore usually set to listen on a small number of ports. Sometimes only a single port is used. This is often the case in reality, where vulnerabilities are typically associated with specific applications, and these applications often run on a single well-known port.

When the worm becomes active, on each timestep it sends as many attack packets as its rate allows. Each packet is sent to a randomly selected address and port within the initialization parameters. The packet is crafted to simulate *exploitation* of a randomly selected vulnerability from its arsenal.

The designed format of the packet's payload is:

`EXPLOIT:[active vulnerability #]:[malicious code]`

where the *malicious code* section simulates the effect of sending a malicious binary. As to how the effect is achieved, the malicious code section includes a Java class name, the definition of which is presumed to be accessible by the simulation. It is a subclass of the abstract Payload class, which specifies one method that must be implemented: `execute()`. This method is called by the InsecureProcess if it is successfully exploited. The user can define new Payloads with practically boundless

possibilities. For the `WormProcess`, the `Payload` is a `WormInstaller`. It has the sole purpose of installing a `WormProcess` on the host `Node`.

When an `InsecureProcess` receives an attack packet, it first determines whether the active vulnerability is one to which it is exposed. If so, the exploit is successful with a certain probability. The probability is pulled from a map indexed by vulnerability number. Typically, this value is 20%. In any case, it simulates the uncertainty in a real-world attack surface caused by patching, firewall signature updates, and other configuration changes.

3.4.5.5 Simulated Malicious Traffic Extensions.

The three types of malicious activity described so far (DDoS, scans, worms) may be combined for yet greater impact. For example, a worm may spread much more effectively and stealthily if a scan is conducted first. Similarly, a successful DDoS attack may require less packets if the attacker switches tactics from trying to flood inbound links to trying to flood listening processes instead. Finally, worms may switch from spreading more copies of itself to launching DDoS attacks (or the spread of the worm itself may qualify as a DDoS attack). Such combinations are generally employed by *botnets* [161]. Neither botnets nor any lesser combination is employed in the present research, but such are natural extensions requiring no changes to our network simulation architecture and should be explored in future research.

3.4.6 Interfacing with MASON.

A few details clarify how MASON interfaces with and drives our implemented network simulation.

The components of MASON of primary interest include the `SimState` class, the `Schedule` class, and the `Steppable` interface. The relationship of these components

to MFIRE is depicted in Figure 6:

- The `MF_Network` class is a subclass of `SimState`, and is itself an abstract class. Calling `doLoop()` on any concrete realization of `MF_Network` invokes MASON's simulation execution Process.
- A `Steppable` is an event, or an object that needs to take action at a specified time.
- The `Schedule` class holds `Steppables` and executes them at the scheduled timestep via each `Steppable`'s `step()` method. `Steppables` that fire on the same timestep may be further grouped into a hierarchy of priorities. `Steppables` with the same priority execute in random order.

3.5 MFIRE (Multi Agent System) Design

Having completed the design of the simulation framework, the simulated network's topology generation, components, and dominant traffic-generating processes, what remains is the multi agent system charged with distinguishing benign traffic patterns from malicious. In the following sections, we present the MFIRE architecture - control, communication, and mobility.

The collective activity of the population of agents is tied together at the multi agent system (MAS) level through a controller, which processes the classification decisions ('votes') of individual agents and reports the majority result. Prior to classification, agents may receive sharing assignments from the controller, and share feature values accordingly. Each agent is then able to make a classification based on local as well as shared feature values.

The controller stores agent reputations. Each round, it calculates a rating for each sharing assignment an agent was given. The rating depends on a heuristic measure of

Algorithm 1 Reputation Calculation

denote classification by agent a_j at time t using only local feature values as l_{jt}
denote classification by agent a_j at time t using combined local and shared feature values (e.g. from peer agent a_i) as c_{jt}
denote the majority classification at time t as m_t

Require: $0 < decay \leq 1$

procedure CALCULATE REPUTATIONS($decay$)
 for all agents a_i **do**
 for all recipients a_j of information provided by a_i **do**
 if $c_{jt} = m_t$ **then**
 if $l_{jt} = m_t$ **then**
 $rating_{ij} \leftarrow neutral$
 else
 $rating_{ij} \leftarrow positive$
 end if
 else
 if $l_{jt} = m_t$ **then**
 $rating_{ij} \leftarrow bignegative$
 else
 $rating_{ij} \leftarrow smallnegative$
 end if
 end if
 $reputation_i \leftarrow reputation_i + rating_{ij}$
 end for
 $reputation_i \leftarrow reputation_i \times decay$
 end for
end procedure

how much the shared feature values helped or hurt the recipient’s ability to classify in step with the majority. After all ratings are processed, the controller may then decay each agent’s reputation by 10%. The idea is to motivate agents to explore other nodes when they are not perceived as making any positive contributions to the community. We experiment with and without this decay.

Algorithm 1 details the idea. The values for variables *neutral*, *positive*, *bignegative*, and *smallnegative* are reflected in Table 2 and discussed in Section 3.5.5.

The use of a centralized controller makes the multi agent system more vulnerable to disruption, but simplifies the design considerably. The disadvantages of a centralized controller may be remedied by allowing agents to elect a new controller when the current controller becomes unresponsive. This is not currently implemented and is also an area for further research.

3.5.1 MFIRE Design Operational Objectives.

From sections 1.4 and 1.3, design objectives for our multi agent system include:

- Minimize classification error
- Minimize bandwidth
- Provide a robust communications protocol to cope with disruptive traffic patterns

See Section 3.2 for formalizations of the first two objectives. These competing objectives are achieved by finding a ‘good’ distribution of a small population of agents. The agents are mobile, and their locational stability is governed by a centralized reputation system managed by the agent controller.

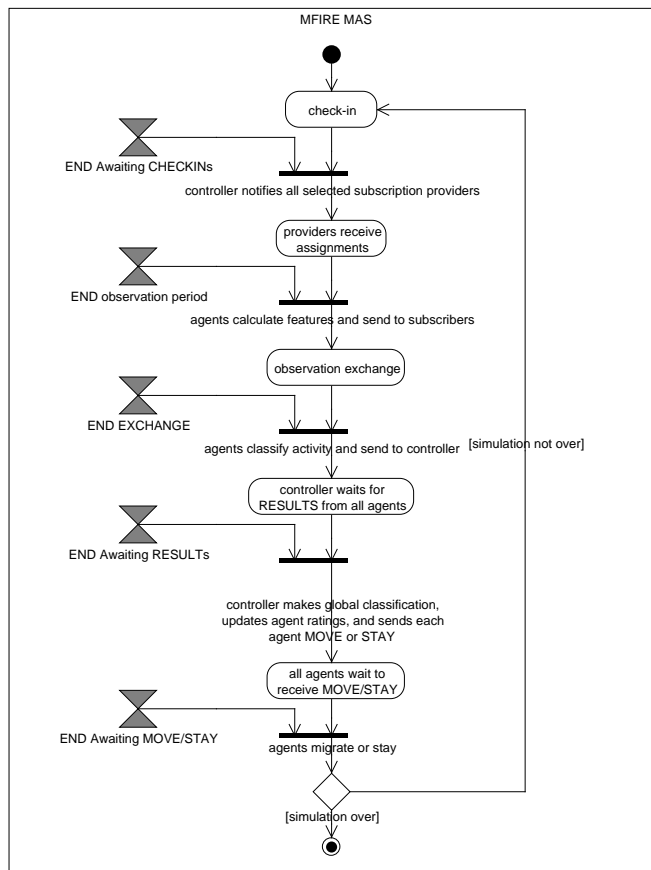
The third objective is elaborated following an overview of the MAS flow of execution.

3.5.2 MFIRE Execution Flow Design.

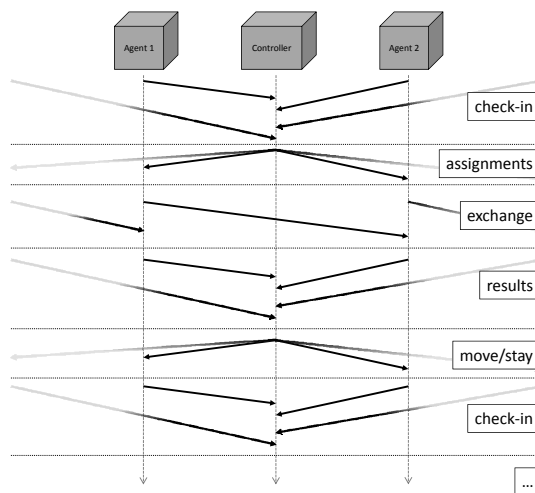
Figure 10 presents a high-level view of the nominal flow of execution from the perspective of the MAS. Five states are shown. Figure 10a indicates that the transition from each state is governed by the clock. This implies synchronization among participating elements, which is discussed in detail in Section 3.5.3. Typical message exchange for each state is shown in Figure 10b.

The explanation of MFIRE’s high-level states is made simpler by assuming agents have been collecting observations from their respective host nodes for nearly a full cycle when it comes time to check in with the AgentController. Furthermore, each agent is assumed to have a reputation stored with the AgentController.

- Check-in: Agents notify the AgentController of their intention to participate in the next round of observation exchange and classification. The AgentController notes the source address and port of each CHECKIN message.
- Transition: The AgentController makes an observation sharing assignment for each Agent that checked in. It does this by constructing a roulette wheel from the reputations of other checked-in Agents. This roulette wheel is used to make a sharing assignment stochastically with preference given to Agents with higher reputations. The AgentController notifies the selected Agent with an ASSIGN message.
- Assignments: Selected Agents receive assignments. Some Agents may receive multiple sharing assignments, while others receive none. For each assignment received, the Agent stores the address and port for the designated recipient as contained in the ASSIGN message.
- Transition: End the current observation cycle, calculate features, and start a new observation cycle. Observations are traffic statistics collected on each



(a) Activity Diagram



(b) Communications Example

Figure 10. MFIRE activity and client-server diagrams showing the system's normal flow of execution

timestep. At the end of each observation cycle, there exists an Observation set for each traffic statistic measured. Features typically summarize one or more of these Observation sets. Agents calculate Feature values and store them for later use. Any Agents with sharing assignments also send their set of Feature values to all assigned recipients using SHARE messages.

- Observation Exchange: Agents wait to receive SHARE messages. Each Agent expects to receive one.
- Transition: Agents use two classifiers to make two classifications for the network activity observed over the previous cycle. One of these uses only locally calculated feature values, while the other uses the combined set of local and received feature values¹. Agents send the results to the AgentController in a RESULTS message.
- Results: The AgentController receives RESULTS messages from all checked-in Agents.
- Transition:
 - The AgentController tallies the votes. In each RESULTS message, the vote is the classification made using the combined local and shared feature value sets. When this is not available because the Agent never received a SHARE message, the AgentController uses the classification made using only the local feature value set, weighted for less influence. The system’s classification is the majority vote. See Algorithm 2, in which θ_l represents the weight of a classification derived from local feature values only.
 - The AgentController updates each Agent’s reputation. For each Agent, each sharing assignment it had garners a rating which can positively or

¹Feature definitions are discussed in Sections 3.5.4.1 and 3.5.4.2.

Algorithm 2 MAS Classification

denote classification by agent a_i at time t using only local feature values as l_{it}
denote classification by agent a_i at time t using combined local and shared feature values (e.g. from peer agent a_j) as c_{it}
denote the majority classification at time t as m_t
denote network activity classes as $\mathcal{A}_k \in \mathcal{A}$ for $1 \leq k \leq K$
denote the vote tally for network activity class \mathcal{A}_k at time t as v_{kt}

Require: $0 \leq \theta_l \leq 1$

```
procedure MASCLASSIFICATION( $\theta_l$ )  
  for all received RESULTS messages  $results_{it}$  do  
    if  $results_{it}$  contains a combined classification  $c_{it}$  then  
      add 1 to the vote tally  $v_{kt}$  for  $\mathcal{A}_k$  for  $k = c_{it}$   
    else  
      add  $\theta_l$  to the vote tally  $v_{kt}$  for  $\mathcal{A}_k$  for  $k = l_{it}$   
    end if  
  end for  
   $m_t = k : v_{kt} = \max_h v_{ht}$  where  $1 \leq h \leq K$   
  return  $m_t$   
end procedure
```

negatively affect the reputation. Every Agent furthermore has its reputation decayed regardless of whether it had a sharing assignment, and regardless of whether it checked in. See Algorithm 1.

- The AgentController sends each Agent a STAY or a MOVE instruction based on whether the Agent’s reputation is above or below a threshold.
- Wrap-up: Agents wait to receive MOVE or STAY. Upon receiving MOVE, an Agent selects a neighboring node at random and sends a MIGRATE message to the node’s AgentManager.

Figure 11 shows the flow of execution of the Agent and the AgentController independently. From this Figure it can be deduced that the AgentController has merely two states: it is either waiting for Agents to check in, or it is waiting for the Agents to send their results, with significant actions taking place on the transitions between states as described above. Meanwhile, the Agent has a collection of synchronization-

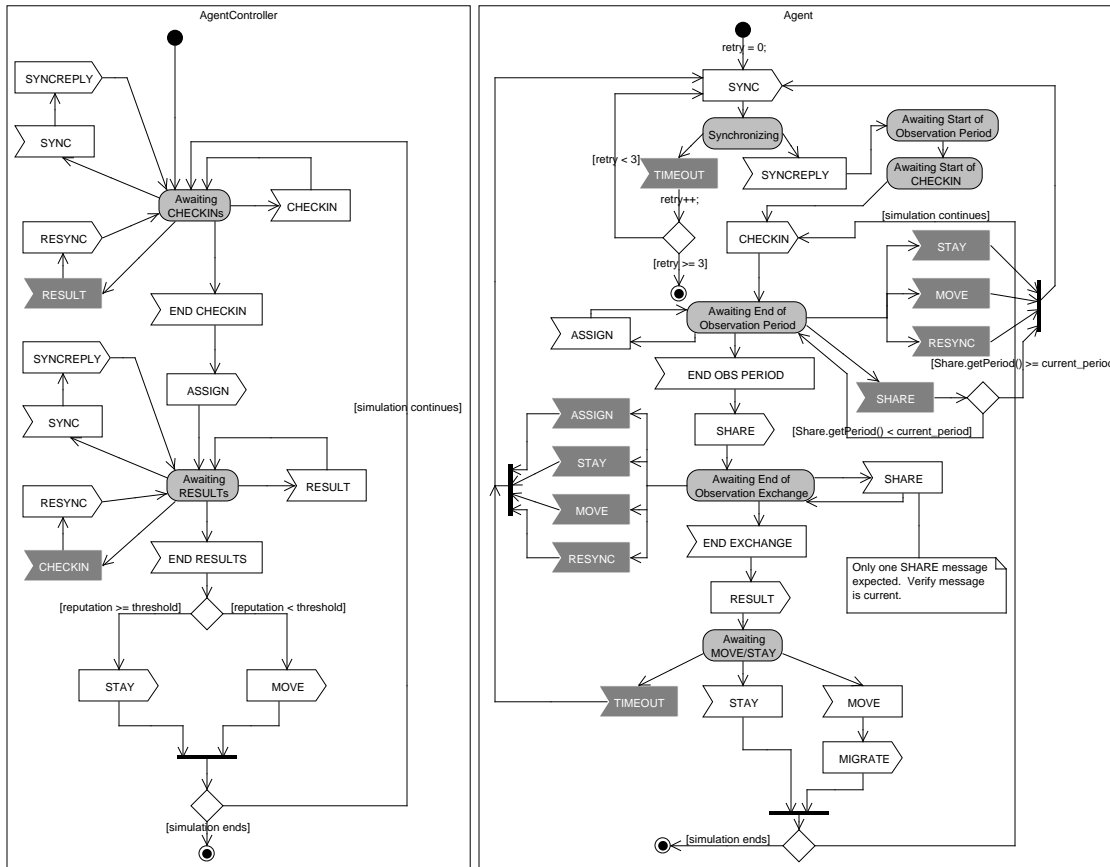


Figure 11. MFIRE detailed activity diagrams for the controller and the agent

related states, and three of the nominal states described above. It is either waiting for an ASSIGN message from the AgentController, or it is waiting for a peer to send a SHARE message, or it is waiting for a MOVE or STAY message from the AgentController.

3.5.3 MFIRE Robust Communications.

With each message, the sender and recipient need to be in agreement with respect to the higher-level flow of execution. The sender thus includes a timestamp in the message indicating the expected time at which the recipient's current state should end. The type of message and the timestamp are both used by the recipient to determine whether the message is 1) expected and 2) current.

It may happen that packets are delayed or lost due to congestion on the network. Therefore, in each state, the Agent or AgentController needs to be able to handle packet loss as well as the arrival of outdated or otherwise spurious messages. As evidenced by Figure 11, unexpected messages and timeouts generally trigger a resynchronization.

Synchronization is a process whereby the Agent can update its clock to match that of the AgentController as well as determine the schedule indicating the time of future state transitions. This allows the Agent to start collecting observations at the same time as others in the system, check in with the AgentController at the right time, and so forth.

Clock synchronization uses the same basic algorithm employed by the Network Time Protocol (NTP) described in RFC 5905 [118]. The Agent sends a SYNC message to the AgentController. The AgentController responds with SYNCREPLY, which contains the time it received the SYNC message as well as the time it sent the SYNCREPLY.

The Agent determines the offset by which it needs to adjust its internal clock to match that of the AgentController:

$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2} \quad (16)$$

where t_0 is the time, by the Agent’s clock, at which it sent SYNC, t_1 is the time, by the AgentController’s clock, at which the SYNC was received, t_2 is the time SYNCREPLY was transmitted (also by the AgentController’s clock), and t_3 is the time, by the Agent’s clock, at which SYNCREPLY was received. This synchronization works well so long as the traffic in each direction has symmetrical nominal delay. Otherwise, a bias is incurred of half the difference between the forward and backward travel times [64].

For details on each type of message that is sent by the system, see Appendix B.

3.5.4 MFIRE Agent: Classification.

In this section we describe the components involved in the Agent’s classification process: the observations and features used, feature selection, and the classification algorithm.

3.5.4.1 MFIRE Agent: Observations and Associated Features.

Each timestep, the Agent queries the host for single-step traffic statistics called *observations*. In pattern recognition terminology, an observation is simply a special type of feature. The connotation is that it is among the most “raw” of features available, in that it takes values directly from the sensors. After a complete observation cycle, the Agent calculates *feature* values over the set of observations. The features used in MFIRE are each based on a single type of observation, and are either the average or the standard deviation of the values reported for that observation over

Algorithm 3 Calculate Features

denote the average of the elements of a set X as $\text{avg}(X)$
denote the standard deviation of the set X as $\text{stddev}(X)$
denote the start of the previous observation period as t_p
denote the start of the current observation period as t_c
denote the observation o_i collected at time t as o_{it}

```
procedure CALC-FEATURES
  for all observations  $o_i : 1 \leq i \leq 14$  do
    for all  $t : t_p \leq t < t_c$  do
       $O_i \leftarrow O_i \cup o_{it}$ 
    end for
     $f_{\text{avg}}(i) = \text{avg}(O_i)$ 
     $f_{\text{stddev}}(i) = \text{stddev}(O_i)$ 
  end for
end procedure
```

the full observation cycle. Infinitely many other features are possible, with likely alternatives including minimum and maximum values observed during the observation cycle, or composite features involving different types of observations. Exploration of an extended feature space including these and more is reserved for future research.

Observations at each node in MFIRE involve sums and ratios for node addresses, ports, packet sizes, and so forth for the full set of currently inbound packets. For example, one of the observations simply counts the total number of inbound packets. Another calculates the average packet size per destination $\langle \text{address}, \text{port} \rangle$ -tuple. For a full accounting of the fourteen observations used in MFIRE, see Appendix B. The selected observation definitions represent statistics of traffic whose distributions could be expected to change as a result of anomalous network activity. For details regarding the calculation of 28 features derived from these observations, see Algorithm 3.

The remaining question concerns the length of the observation period over which features are defined. The length of the period depends on the chosen scale interpretation of the network. As mentioned in Section 3.4.2, three scales are available:

LOCAL, REGIONAL, and GLOBAL. The scale selection impacts the range of link lengths, respectively 1, 1 to 10, and 1 to 100 units. This in turn affects the time required for communications to traverse from one node to another in the network. The duration of the observation period is the cumulative duration of the five phases involved in the MAS flow of execution (see Section 3.5.2 and Figure 10). The duration of each phase is equal with the exception of the CHECKIN phase, which is allotted extra time to allow migrating agents in the previous MOVESTAY phase to complete their migrations before checking in with the AgentController. The duration of each phase is specified according to the longest observed message transit time (in simulation timesteps) from one node to another in a typical 100-node network produced by the selected topology model (see Section 3.4.3), with approximately 20% more timesteps added to accommodate messages reasonably delayed by network congestion. The result is that for the LOCAL, REGIONAL, and GLOBAL scales, the total observation periods are respectively 88, 137, and 813 timesteps long.

3.5.4.2 MFIRE Agent: Combined Features.

The classification using only local features is straightforward. When feature values shared from a peer are also available, the question is how to combine them. Any conceivable binary operation may be applied, and a search for one that produces higher performance falls within the purview of feature generation (see Section 2.2). We consider three for each single-node feature:

1. The absolute value of the difference between the local value and the peer-provided value
2. Their average
3. Their product

3.5.4.3 MFIRE Agent: Feature Selection.

A filter method based on Bhattacharyya coefficient analysis (see Section 2.2.1) ranks the features according to a measure of how well they distinguish each class from the others.

The top three features² from this Bhattacharyya distance-based ranking are preferred. Additionally, we examine the correlation between candidate features. When a pair of features exhibits strong correlation, one of the two is rejected because it is unlikely to contribute useful information beyond what the retained feature provides [70]. MFIRE requires the analysis be done for the local feature set as well as for the combined feature set.

3.5.4.4 MFIRE Agent: Classifier.

The classifier is a Support Vector Machine (SVM) (see Section 2.2 and [91, 74]). From decision trees to linear classifiers to neural networks, there are many techniques to choose from. SVM is selected due to its “high generalization performance without the need to add *a priori* knowledge,” even in the face of many features [34].

Note the Support Vector Machine employs numerical optimization methods to establish the classification model. This process is computationally intense. Future work should identify alternative classification techniques that achieve similar performance with less computational expense. The object-oriented MAS architecture design accommodates any classification technique in the place of the Support Vector Machine (see Section 3.6), or even a mixture of classifiers distributed amongst the agents in the MAS.

²Three features permit visual inspection, by a human, of resulting sample plots of training data sets, to evaluate class separability.

Table 2. How the AgentController Rates Providers of Shared Feature Values

Receiver’s classification result, based on feature sets used

Local Only	Local + Shared	Rating
Same as majority	Same as majority	+0
Same as majority	Differed from majority	-0.1
Differed from majority	Same as majority	+0.1
Differed from majority	Differed from majority	-0.05

3.5.5 MFIRE Reputation.

MFIRE employs a centralized reputation system per the broad categorization of [80] (see Section 2.6). This approach puts the reputation of each agent under the control of a central reputation manager. It allows simplified management of agent interactions, but is prone to single point-of-failure issues. Future research will examine the value of a distributed approach.

Agents start with a base reputation value of 0.5, which is twice the migration threshold value of 0.25 used in experimentation. The AgentController uses Table 2 to modify reputations according to how well providers’ observations helped receivers vote in step with the majority.

When agents vote in step with the majority, and would have done so even without the use of the shared observations, there is no reason to rate their providers positively or negatively. On the other hand, if the agent is prepared to vote in step with the majority, but ends up not doing so due to the influence of the shared observations, the judgment of the crowd is viewed as superior to the opinion of a single peer and thus the provider is rated negatively. Real benefit is perceived when they would have voted out of step with the majority but for the “corrective help” of the shared observations, and in such cases providers are rated positively. If the agent votes out of step with the majority and would have done so even without the shared observations, the provider is rated negatively. But, not so much as if the shared observations had dissuaded the

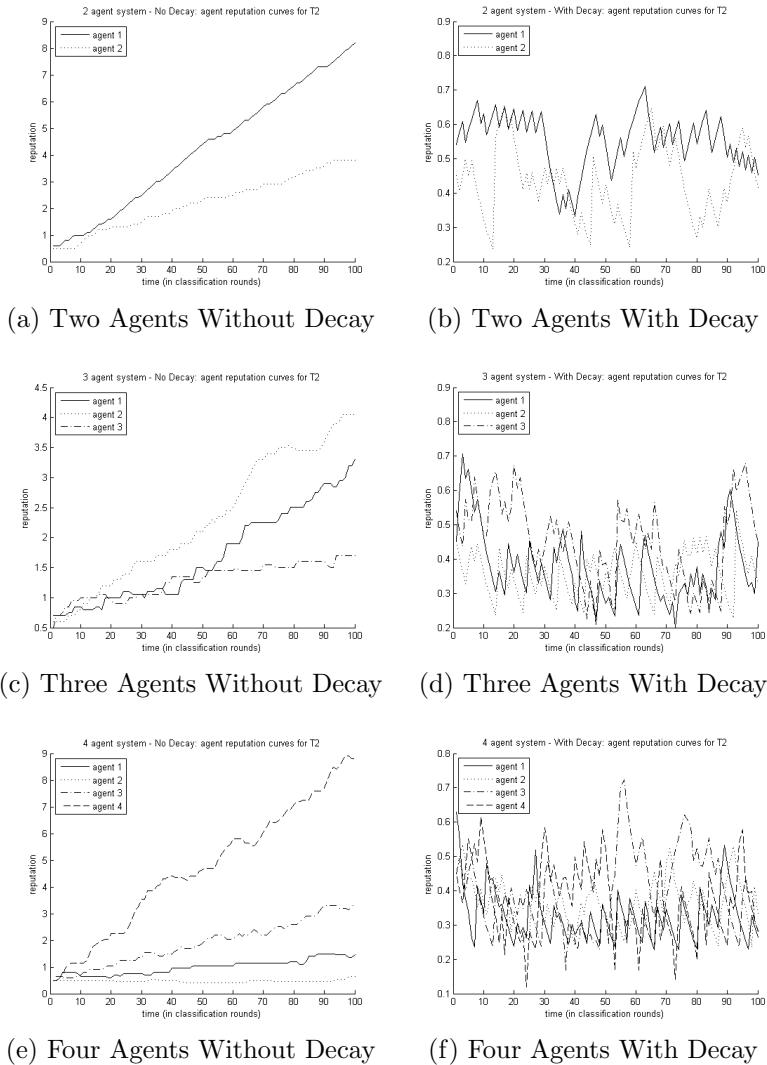


Figure 12. Comparison of reputation curves, using decay vs. not using decay, for single test runs in the MASNAC system

agent from otherwise voting in step with the majority.

Note that reputation is not explicitly provided either a lower or an upper bound. On the lower end, consider an agent whose reputation is at 0.30 just prior to a reappraisal. The lower bound is then $0.30 - 0.1 \times (z - 1)$ where z is the number of agents in the MAS and 0.1 is the magnitude of the most negative rating attainable in Table 2. This reputation value does not last longer than is required for the agent to migrate to new node and receive a reputation reset.

On the upper end, when decay is not used, that is, $decay := 1$ in Algorithm 1, it is often the case that reputation monotonically increases for certain agents in the population. Under this condition, an agent can accrue a sufficiently high reputation as to ensure it is the dominant preference of all peers for the selection of shared information provider. With its feature value sets in high demand, it single-handedly exerts a great deal of influence on the majority classification result, which garners positive ratings (or at least avoids negative ratings) and further reinforces its continued reputation dominance.

This is a major motivation for applying reputation decay. In MASNAC, the predecessor to MFIRE, reputation is applied conceptually the same way. The performance evaluation of MASNAC yields insights into the effect of decay on agent reputation bounds. As demonstrated in Figure 12, setting $decay := 0.9$ keeps agent reputations bounded below 0.8 for the test runs shown.

A final note before proceeding to the details of MFIRE agent mobility: Table 2 shows that rating values are the same regardless of whether the MAS classification results from a 51% or a 99% majority. Future work should investigate the benefit of incorporating the size of the majority into the rating determination.

3.5.6 MFIRE Agent Mobility.

Recall that when an agent's reputation drops below a threshold, the AgentController sends a MOVE message during the wrap-up phase (see Figure 11). The threshold value used in experimentation is 0.25. In any case, upon receipt of a MOVE message, the Agent sends a MIGRATE message to a randomly-selected neighboring node.

Specifically, the message is sent to an AgentManager on that node. An AgentManager is installed on every node on a common port specified in the PortDirectory.

The MIGRATE message contains the Agent’s state, allowing the remote Agent-Manager to re instantiate the Agent. An Agent’s state consists of its classifier, observations used, features used, and any required initialization parameters. Once the AgentManager receives the message, it sends MIGRATEACK to the AgentManager at the original node. This AgentManager terminates the original executing copy of the Agent. In Java terminology, it removes the only existing reference to the Agent (held by the Schedule), making it eligible for garbage collection. The re instantiated Agent’s first action is, of course, to synchronize with the AgentController and resume participating in the MAS at the earliest opportunity.

3.5.7 Agent Distribution: Evidence of Self Organization.

The ability of MFIRE to find effective agent distributions (in terms of location) without external influence is evidence the system possesses attributes of self organization [41, 71, 128]. As summarized by Dempster: “Self-organization refers to exactly what is suggested: systems that appear to organize themselves without external direction, manipulation, or control” [43].

3.5.8 Stability: An Emergent Property.

As agents find ‘good’ vantage points, resulting in consistently high reputation, mobility decreases, which is to say, stability increases. Because the objective of stabilizing the mobility of the agent population is not modeled or explicitly sought by individual agents, this qualifies as an emergent property of the multi agent system (see Section 2.8 and [41], [71], [128]).

3.5.9 Optimization via a MOEA.

There are a variety of parameters to explore when considering how to optimize the multi agent system with respect to minimizing classification error. One way of fine tuning the system is to use a multi-objective evolutionary algorithm (MOEA), as described in Section 2.7. For example, a MOEA can find values for the rating table (replacing those shown in Table 2) that result in ‘good’ classification performance across all traffic classes, based on improvement observed in the average classification accuracy in the second half of each test run compared to average classification accuracy in the first half.

This multi objective approach contrasts with equation 7, which calls for a loss-function that returns a single value. In this second iteration of our research efforts, equation 7 is the general approach to training the classifier employed by the agents. The specific classifier used, the Support Vector Machine, is described in Section 2.2. The higher-level MAS classifier, on the other hand, maintains the objectives separately, where each objective is the minimization of prediction error for a specific class. That is, there are K objectives when there are K classes, and each objective $f_k(X)$ is:

$$f_k(X) = 1 - P(\hat{A}(X) = k | A(X) = k) \quad (17)$$

where \hat{A} is the MAS classifier, and $A(X)$ is the true attack class.

We use jMetal ([47]). From Table 1 in Section 2.7.2, jMetal provides the necessary features for our research: it handles continuous as well as combinatorial problems, is an open framework, and is implemented in Java. The Java implementation facilitates integration with MASON and MFIRE.

jMetal furthermore has implementations for a variety of MOEAs. Two of the most popular MOEAs are the Non-Dominated Sorting Genetic Algorithm II (NSGAI)

and the Strength Pareto Evolutionary Algorithm 2 (SPEA2). They differ in the way members of the population, each representing a solution, are ranked after fitness evaluation. See Section 2.7 and [39] for details.

3.6 Java Implementation

Throughout this chapter, the high-level and low-level designs of MFIRE and its network simulation environment are presented with the implication that the implementation follows the design closely. This section makes a few remarks regarding where the implementation includes some additional elements or differs in some way from the low-level due to Java OOD constraints. Unless otherwise noted in this section, the implementation matches the design presented in previous sections.

For example, the package hierarchy design shown in Figure 5 is, for the most part, implemented directly. An additional ‘experiments’ package supports scripted data generation and experimentation.

The low-level design approach that follows demonstrates good software engineering practices.

Apart from simply portraying a lower level of detail, Figure 6 reveals a design refinement over Figure 5. The *Process* class, acting as a supertype, allows many subclasses to be implemented that work seamlessly without requiring changes to the rest of the simulation environment. Note that the *Agent* and *AgentController* classes are shown in Figure 6 to be *Process* subclasses, because they require the same communications facilities the *Process* class provides. Our research implements Figure 6, reflecting a desire to make the system extensible in terms of Processes.

For the same reason, network scenarios are implemented as subclasses of *MF_Network*. More precisely, most of the scenarios are subclasses of *TopgenNetwork* (see Section 3.1). In any case, scenarios have been implemented for each type of malicious activ-

ity investigated (DDoS, scan, worm), and the architecture supports easy expansion to many more scenarios, including combinations of currently implemented malicious behaviors or new behaviors. The fact that `TopgenNetwork` is implemented as a subclass of `MF_Network` means that it can also be swapped for a different subclass that generates topologies in a different way while maintaining interoperability with the rest of the system.

For malicious processes simulating exploitation, such as the `WormProcess`, the *Payload* abstract class is useful. Subclasses (such as the *DoSInstaller* and the *WormInstaller* in Figure 5) simply need to implement the `execute()` method to be ready for use by the simulation.

Though not shown in Figure 6, the same extensibility applies to the software classes involved in classification. Agent methods call methods defined in the *Classifier* class, which is abstract like the *Process* class. In object-oriented languages, abstract classes cannot be instantiated. They specify an interface and provide some common functionality (method implementations) and attributes to any implemented subclass. Thus, the SVM Classifier is implemented as one subclass of *Classifier*, but any other classification technique may be implemented as well, with no changes to the existing design required.

The same holds for the *Observation* and *Feature* classes. Sections 3.5.4.1 and 3.5.4.2 describe implemented observations and features, but these are simply subclasses of abstract classes. Thus, many other features and observations may be implemented for future research.

With respect to the SVM classifier employed locally by each agent, the implementation used is Soergel’s “jlibsvm” [137], a Java port of LIBSVM [31]. The common implementation programming language simplifies integration with MFIRE.

With respect to visualization of the simulation (see Sections 3.1 and 3.4 as well

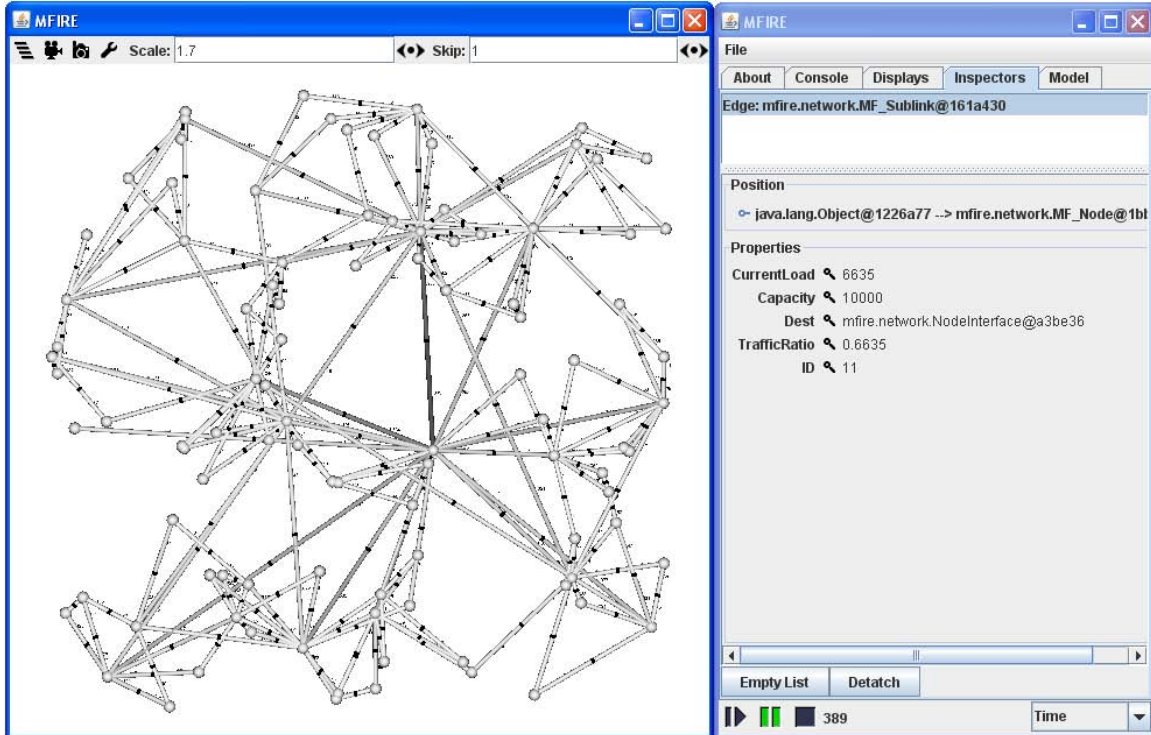


Figure 13. MFIRE network simulation environment using MASON’s visualization and GUI facilities

as Figure 5), we implement *MF_ModelWithUI* as a subclass of MASON’s *GUIState*. This provides access to MASON’s visualization and GUI facilities (see Figure 13). *MF_ModelWithUI* is supplied an *MF_Network* in the constructor. From this *MF_Network* (e.g. a *DDoSTopgenNetwork*, *ScanTopgenNetwork*, etc.), the *MF_ModelWithUI* extracts information required for visualization and presents it using separately defined classes that inherit from one of MASON’s *Portrayal* classes. As shown in Figure 5, implemented *Portrayals* include the *NodePortrayal* and the *LinkPortrayal*. These classes use public methods in their associated domain classes to retrieve any information required for visualization. For example, the *LinkPortrayal* adjusts the color of the visualized *Link* in accordance with the *Link*’s current traffic load.

A small but significant detail for visualization is the use of intermediate timesteps to smooth the animation; specifically, the *Link* color transitions. We insert a number

of intermediate visualization timesteps between each pair of consecutive domain simulation timesteps. On each visualization timestep, the current Link color is updated to a shade closer to the ‘target’ color for the Link’s current load. Typically, we use either 10 or 100 visualization timesteps, depending on whether animation smoothness or simulation speed is more important. This approach could allow animation of key packets transiting the network, especially those used for agent migration or worm propagation. Such animations will yield faster intuitive insights in future research. In current research, the changing colors of the Links aids visual validation of desired simulated traffic effects, such as those caused by DDoS (see Section 5.2.2.1, Figure 17) and worm attacks (Section 5.2.2.3, Figure 20).

3.7 Summary

This chapter covers the aspects of the MAS-network system design, including the network simulation framework and implementation of the multi agent system for network attack classification. We describe how the MAS is designed to show evidence of self organization as well as emergence. Finally, the MOEA approach is discussed for finding rating table values that result in good MAS classification performance for each network attack class.

The next chapter presents the major differences between MFIRE and its predecessor, MASNAC, and their associated network simulation environments.

IV. MASNAC: Network Simulation and Multi Agent System Design

MFIRE (and its network simulation environment), presented in Chapter III, represents the second design iteration of our development. In this chapter, the initial MASNAC design is presented. It provides an important understanding of the associated rationale for the design evolution into MFIRE.

This chapter summarizes key differences between the first and second iterations of our research. Section 4.1 presents an overview. Section 4.2 identifies key differences in the network simulation. Section 4.3 focuses on the differences in the multi agent classification systems. Section 4.4 summarizes the chapter.

Though essential system validation tests for MFIRE and its network simulation environment are complete and presented in Section 5.2.2, total classification performance evaluation for MFIRE awaits further development. In the case of MFIRE's predecessor, MASNAC, however, classification performance evaluation is reported in Section 5.2.1 and [73] and [72].

4.1 Overview: Key differences

Table 3 summarizes the key differences between the first and second iteration of our research. In general, the second iteration represents a significant step up in terms of model and multi agent system complexity. There are also some differences in some of the key assumptions.

4.2 Network Simulation

This section focuses on the specific differences in the network simulation used in each iteration.

Table 3. Comparison of Iterations 1 and 2

	Iteration 1	Iteration 2
AS Network Scale	‘local’ only	‘local’, ‘regional’, ‘global’
AS Network Size	10 nodes	100 nodes
AS Network Topology	manually designed	produced by Internet topology modeler
Node Behavior	restricted processing capacity / shut down under heavy load	unrestricted processing capacity
Packet Payloads	simulated quantity only	payloads implemented and used for interprocess communication
Attacks	DoS	DDoS, Worm, Scan
MAS classifier	minimum euclidean distance	support vector machine
MAS communications	out-of-band, instantaneous	in-band with network-based delays
Feature Selection	wrapper method in a MOEA	filter method: Bhattacharyya distance, correlation
MAS Objective	identify source and target of DoS attack	identify type of attack

4.2.1 Node Behavior.

One of the major differences in key assumptions relates to the ability of nodes to handle large quantities of traffic. The first iteration network simulation sets a per-node traffic processing limit equal to 80% of the aggregate inbound link capacity. Once this quantity is exceeded, the node is ‘knocked off-line’, muting all resident processes and rendering connected links unusable. The second iteration removed this effect due to a lack of a strong modeling basis for representing load-based router failure conditions. Future research may reinstate the router failure effect, as it is observed in the real world (see [32] for an example).

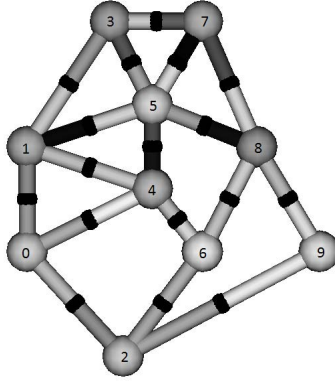


Figure 14. Iteration 1 network

4.2.2 Size and Scale.

Other network simulation differences relate to size and scale. Figure 14 shows the network that served as the basis for MASNAC performance evaluations. It consists of 10 nodes compared to the 100-node networks used in MFIRE development and testing. Furthermore, first iteration networks consist of links that are all one unit in length. The second iteration network design is much more permissive, including three different scales, each with a different range of link lengths, to accommodate clients with different needs (see Sections 3.4.1 and 3.4.2).

4.2.3 Topology.

The topology of the network shown in Figure 14 is manually designed. In MFIRE's network design, by contrast, topologies are generated by the Topgen Internet topology modeler (see Sections 3.4.3 and 2.1.2) incorporating the latest in Internet topology modeling techniques.

4.2.4 Packet Payloads.

The packets in the MASNAC's network design do not carry any usable content. A 'quantity' field simulates the size of the packet and is used to create the desired

Table 4. Attack Classes

Class	Source	Target
1	0	7
2	9	7
3	5	7
4	0	3
5	9	3
6	5	3
7	0	2
8	9	2
9	5	2
10	-	-

traffic-based effects. Packets in MFIRE networks may be created that behave the same way, or one can create packets that carry actual messages used for simulated interprocess communication across the network.

4.2.5 Attacks.

The attacks implemented for the first iteration network simulation are all of the simple Denial of Service (DoS) variety. Each DoS attack is defined by a source and a target. In each case, the technique is the same: send large numbers of small packets from the source to the target in an attempt to flood all connected links. The combinations used in experimentation are listed in Table 4.

4.3 Multi Agent System

This section focuses on the multi agent system aspect of each design iteration and how they differ. Both systems make use of a reputation system governing agent mobility. But the classifiers, communications, features, feature selection methods, and classifier objectives all have some key differences.

4.3.1 MASNAC Classifier.

The MASNAC classification technique is the simple minimum distance classifier (MDC), compared with MFIRE’s support vector machine (see Section 2.2 for descriptions and references for both).

MASNAC, like MFIRE, has agents share observations with each other. Agents make two classifications: one based purely on locally-available observations and one based on both local and remotely provided observations. As in MFIRE, the classification from the combined set of observations is used to represent the vote of each agent as to the attack class, with the majority vote used as the system’s output.

4.3.2 MASNAC Communications.

MASNAC communications are assumed to be out-of-band, not subject to the hazards of congestion in the network. Communications are furthermore instantaneous; neither propagation nor transmission delay are modeled.

MFIRE, on the other hand, is subject to the potential difficulties arising from network congestion: packets sometimes fail to arrive, and MFIRE is designed to cope with this situation. Even without congestion, normal transmission and propagation delays affect MFIRE communications.

4.3.3 MASNAC Features.

Features in MASNAC classification are all node-specific. Each feature is furthermore a statistic related to the traffic quantity arriving on a particular inbound link and headed for a particular destination node. For each such link-destination pair, we collect traffic quantity statistics (mean and standard deviation) over ‘short’, ‘medium’, and ‘long’ periods of time.

In order to fully characterize each attack class for classifier training purposes, it

is necessary to collect these statistics for every node in the network. Each agent is supplied each of the class means in order to employ the minimum distance classifier. Note that a feature subset selection string produced by the feature selection process (see Section 4.3.4) restricts the actual features used in the MDC.

This requirement to collect all statistics for every node in the network has significant implications for the scalability of MASNAC and motivates a change in how features are defined for the MFIRE in the second iteration.

MFIRE features are not meant to be associated with specific nodes in the network. Rather, training samples are created by sampling statistics from two random nodes in the network. Candidate nodes in this process are restricted to a subset of the first 20 nodes created by the topology generation process. Due to preferential attachment in the topology generation model, these nodes are more likely to be highly connected compared with nodes created later in the process (see Section 2.1.2 and [138]).

4.3.4 MASNAC Feature Selection.

The agents in MASNAC are supplied a feature subset selection vector which indicates, for every node in the network, which local features are eligible for use in classification.

The feature selection process in MASNAC is handled by the Non-Dominated Sorting Genetic Algorithm (NSGA-II) (see Section 2.7.1 and [39]). For reasonable convergence, parameters for NSGA-II include a population size of 200, a crossover probability of 0.9, a mutation probability of 0.02 (per bit in each chromosome), and an evaluation limit of 200,000 generations.

The chromosome of each individual in the population is a feature subset selection vector S_X as in Section 3.2. The minimum distance classifier is used in the fitness function, which is also supplied with a set of labeled validation data. The problem is

formulated as a multi objective optimization problem, with criteria f as follows for $i = 1, \dots, K$:

$$(\text{minimize}) f_i(\vec{x}^*) = 1 - P(\hat{A}(x) = \mathcal{A}_k | A(x) = \mathcal{A}_k) \quad (18)$$

An additional objective f_v is to minimize the number of distinct nodes providing the features involved in classification. The purpose of this objective is to reduce the bandwidth required for MASNAC's classification process.

As reported in [73], the NSGA-II MOEA typically produces 25 selected features that are somewhat evenly spread across all ten nodes.

4.4 Summary

This chapter discloses key differences between the first and second design iterations of our research. The distinguishing characteristics of the network simulation environments developed in each iteration are provided. The contrast between the multi agent system architectures, their communications and their classification methodologies, is also clarified.

The next chapter provides testing results and analysis: for the first iteration, MASNAC performance is evaluated; for the second iteration, system validation tests demonstrate essential model and MAS functionality.

V. Experimentation and Analysis

The previous two chapters provided system design and implementation details for MASNAC and MFIRE. This chapter presents the experimentation and analysis plan evaluating the hypothesis objectives stated in Section 1.3. Section 5.1 describes the experimental design. Results and analysis are presented in section 5.2. Finally, Section 5.3 concludes the chapter.

5.1 Experimental Design

Barr et al. [16] explain the need for an experimental design that helps determine whether a new heuristic method contributes something important. They present a list of possibilities. A heuristic method makes a contribution if it is:

- Fast: produces high-quality solutions quicker than other approaches;
- Accurate: identifies higher-quality solutions than other approaches;
- Robust: less sensitive to differences in problem characteristics, data quality, and tuning parameters than other approaches;
- Simple: easy to implement;
- High-impact: solves a new or important problem faster and more accurately than other approaches;
- Generalizeable: having application to a broad range of problems;
- Innovative: new and creative in its own right.

Barr furthermore asserts [16] that research reports about heuristics are valuable if they are:

- Revealing: offering insight into general heuristic design or the problem structure by establishing the reasons for an algorithm’s performance and explaining its behavior;
- Theoretical: providing theoretical insights, such as bounds on solution quality

From Section 1.3, the goal of our research is to develop an effective flow-based, multi agent system for inter-AS network attack classification.

The heuristic used to search for effective agent distributions is identified in the hypothesis: that a flow-based, multi agent network attack classifier can be made more effective by:

1. employing a reputation system to govern agent mobility
2. adding a decay factor to each agent’s reputation to further spur agents to find nodes providing the most “useful” information

Therefore, the reputation system is the heuristic under study. Qualitatively, we observe that it is simple to implement and innovative. But the principle contribution from Barr’s list our experimentation aims to demonstrate is that the use of a reputation system increases the accuracy of the multi agent network attack classifier.

The goal of our experimental design is to demonstrate whether we succeed in our research objectives. From Section 1.3, the research objectives are to:

1. *Develop an effective network simulation environment appropriate for the problem scope.*
2. *Validate the proper functioning of simulated malicious traffic.*
3. *Validate the proper command, control, and communications in the multi agent intrusion detection system.*

4. *Study the effects of several factors on classification accuracy.*

The order of these objectives suggests a natural chronological sequence of development and testing. It is the case, however, that MASNAC, the first iteration of design and implementation, is quantitatively assessed in terms of classification accuracy, while its successor, MFIRE, has yet to receive a similarly detailed assessment.

We therefore reorder the research objectives, starting with MASNAC's performance assessment and proceeding to the evaluation of the successor network simulation environment and multi agent system MFIRE. Thus, the refined and renumbered objectives are to:

1. *Study the effects of several factors on MASNAC classification accuracy.*
2. *Develop an effective network simulation environment appropriate for the problem scope and exhibiting more complexity than the environment used for MASNAC.*
3. *Validate the proper functioning of simulated malicious traffic in the second iteration network simulation environment.*
4. *Validate the proper command, control, and communications in the MFIRE.*

The benchmarks for these research objectives are given in the next two sections, which detail respectively the designed performance assessment for MASNAC and the validation for MFIRE.

5.1.1 MASNAC Performance Assessment: Response Variables, Factors, and Statistical Design.

For MASNAC's performance assessment, we use the Wilcoxon rank-sum statistical comparison technique discussed in Section 2.9. We examine the minimum, maximum, mean, and standard deviation in classification accuracy over 30 test runs for each

selected combination of factors. Conducting this many test runs allows a qualified judgment as to whether normality assumptions for the generated distribution are justified. Each test run consists of a persistent instance of MASNAC classifying and adapting over the course of 100 consecutive attack rounds. For each attack round, the attack (see Table 4) is selected uniformly randomly.

The factors studied are:

- Number of agents: two, three, and four
- Reputation: not used, used without decay, and used with decay; denoted in some of the tables respectively as **A**, **B**, and **C**
- Time of MAS classification: four, eight, twelve, and sixteen timesteps into the simulation, denoted T1, T2, T3, and T4

Note that when reputation is not used, on each round each agent uniformly randomly selects a peer to provide shared observations. Agents in this case do not move from their randomly-selected starting nodes.

5.1.2 MFIRE: Qualitative Evaluations.

Assessment of the effectiveness of the MFIRE’s network simulation environment includes the following tests, resulting from the research objective and benchmark specification presented in Section 1.3:

- Implement a DDoS attack and a ping process. Use the ping process under benign network conditions to ping the target node. Demonstrate reliable response from the target. Launch the DDoS attack at the same target node. Ping this node during the attack and demonstrate unreliable response from the target.
- Implement a UDP-style scan attack [146] involving a scanning process, an target process that responds actively to scan packets, ‘normal’ processes that make

no response, and ICMP “port unreachable” messages sent by the host node in response to packets sent to ports with no listening processes. Set up some target processes on random ports on selected nodes. Launch a block scan covering the possible addresses and ports where the target processes might be found. Print the scan report once the scan is complete (each sent scan packet resulted in either a response or timeout). Demonstrate ability to find target processes via scanning.

- Implement a worm process and an insecure process. Simulate a global set of vulnerabilities and associated exploits. Initialize a worm processes with a subset of exploits. Initialize insecure processes all over the network, each with a random subset of vulnerabilities. Let the worm loose on one initial node. Observe the worm’s ability to propagate copies to other nodes when it contains an exploit matching a vulnerability held by a resident insecure process. Observe new worm attacks spreading from newly-exploited nodes.

Assessment of the command, control, and communications of the second iteration’s multi agent system include a series of tests validating proper flow of execution as designed, as shown in Figure 11. The full set of tests for MFIRE execution flow validation is provided in Appendix C.

5.1.3 Test Computational Environment Factors.

Experiments are conducted on an HP EliteBook laptop, 2.79 GHz Intel Core 2 Duo processor, 2.96 GB of RAM, Windows XP Professional 2002, Service Pack 3. Java Runtime 1.6 update 22 is used. Simulations and experiments are developed and executed in the Galileo distribution of the Eclipse integrated development environment.

The full set of tests executed and completed within six hours.

5.2 Results and Analysis

The results of experimentation for MASNAC and MFIRE are presented in this section.

5.2.1 MASNAC Performance Assessment: Results.

Several experimental observations can be made from the statistics reported in Tables 5, 7, and 9 providing insight into MASNAC performance.

First, regardless of the number of agents used, and regardless of the timestep at which the classification is made, using reputation generally outperformed the system that did not. Using reputation with decay generally outperforms using reputation without decay. Decay spurs those agents not receiving negative ratings to nevertheless move to different nodes if over time they do not receive positive ratings, either. It is presumed that this slight increase in exploration yields the measured benefits, but Wilcoxon rank-sum hypothesis testing (reported in Tables 6, 8, and 10) shows this cannot be conclusively asserted in all cases.

Second, in most cases classification accuracy starts low at T1, achieves a maximum at T2, and degrades as the time of classification within each scenario is pushed to T3 and T4. This is because as time passes during a denial of service attack, more nodes have the possibility of shutting down, generating cascading effects as packets are routed differently. With a larger window of time, these cascading effects cause the variance in the traffic statistics to grow, making minimum Euclidean distance-based classification more difficult.

Third, the standard deviation trends downward from using no reputation, to using reputation without decay, to using reputation with decay, reflecting potentially more consistent results from the latter system.

In these tables, **bold** indicates the ‘best’ value obtained for the column statistic.

Table 5. Two Agent Classification Accuracy Over 30 Runs

System	Class. Time	Min	Mean	Max	Std Dev
No Rep.	T1	0.25	<i>0.59</i>	0.82	0.14
	T2	0.29	0.63	0.87	0.16
	T3	0.48	0.65	<i>0.81</i>	0.09
	T4	0.38	0.60	0.73	0.08
Rep w/out Decay	T1	<i>0.36</i>	0.56	<i>0.83</i>	<i>0.11</i>
	T2	0.24	0.66	0.86	0.13
	T3	0.50	0.62	0.76	0.08
	T4	0.42	0.58	<i>0.77</i>	0.09
Rep w/ Decay	T1	0.31	<i>0.59</i>	0.80	0.12
	T2	<i>0.44</i>	0.69	0.88	<i>0.09</i>
	T3	0.53	<i>0.67</i>	0.76	0.06
	T4	<i>0.46</i>	<i>0.61</i>	0.73	<i>0.07</i>

Table 6. P Values For Two-Sided Wilcoxon Rank Sum Test for Two Agent System

	Median			P-Value		
	A	B	C	A vs B	A vs C	B vs C
T1	0.600	0.540	0.605	0.166	0.801	0.188
T2	0.635	0.635	0.695	0.469	0.176	0.164
T3	0.670	0.630	0.670	0.280	0.403	0.014
T4	0.600	0.585	0.625	0.374	0.524	0.150

In the ‘Mean’ column only, numbers in *italics* represent the ‘best’ values obtained for a particular classification time across the three reputation approaches.

In Table 5, for the four classification times, four out of four best mean results are obtained by the system using reputation with decay (including a tie at T1 with the system that did not use reputation). Using reputation with decay furthermore achieves three of the four best (smallest) standard deviations, indicating more consistent results.

To validate the conclusion that using reputation with decay is best, we perform

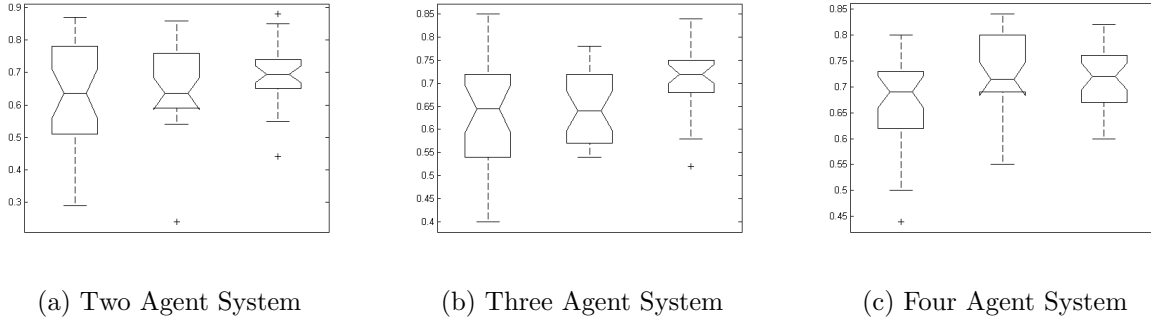


Figure 15. Classification accuracy of MASNAC under different parameters; in each boxplot: no reputation; reputation without decay; reputation with decay

hypothesis testing using a two-sided Wilcoxon rank-sum test. The P-values reported in Table 6 indicate the likelihood that two sets of samples could have been collected under the assumption they came from a single distribution. Low values suggest statistical evidence of significant differences in system performance. Anything under 5% is marked **bold**. The same information is conveyed visually using boxplots in Figure 15.

It turns out that, according to Table 6, despite the case for optimism presented in Table 5, a statistically significant increase in performance is elusive. It is observed in one case for the two-agent system: at T3, using reputation with decay is significantly better than using reputation without decay.

In the three agent case, Table 7 again suggests superior performance when using reputation with decay for T1, T2, and T3. In Table 8, MASNAC using reputation with decay is shown to have significantly better performance than either of the alternatives at T2.

In the four agent case (Table 9), three of the four best mean results are produced by reputation without decay. Reputation with decay achieves the best mean result at T3, and posts the best results for standard deviation in all levels of the time factor. However, Table 10 indicates that it is the system using reputation without decay that

Table 7. Three Agent Classification Accuracy Over 30 Runs

System	Class. Time	Min	Mean	Max	Std Dev
No Rep.	T1	0.34	0.56	<i>0.75</i>	0.10
	T2	0.40	0.63	0.85	0.13
	T3	0.55	0.65	0.77	0.06
	T4	0.46	0.59	0.72	0.08
Rep w/out Decay	T1	0.35	0.56	0.74	0.09
	T2	<i>0.54</i>	0.65	0.78	0.08
	T3	0.51	<i>0.66</i>	0.79	0.07
	T4	0.41	0.60	0.73	0.09
Rep w/ Decay	T1	<i>0.42</i>	<i>0.59</i>	0.71	<i>0.07</i>
	T2	0.52	0.71	0.84	0.06
	T3	0.50	0.64	<i>0.81</i>	0.07
	T4	<i>0.51</i>	<i>0.61</i>	<i>0.74</i>	<i>0.06</i>

Table 8. P Values For Two-Sided Wilcoxon Rank Sum Test for Three Agent System

	Median			P-Value		
	A	B	C	A vs B	A vs C	B vs C
T1	0.535	0.540	0.590	0.853	0.178	0.195
T2	0.645	0.640	0.720	0.679	0.009	0.006
T3	0.660	0.660	0.625	0.584	0.374	0.153
T4	0.575	0.630	0.590	0.433	0.254	0.756

achieves statistically significant better performance than the system not using reputation for T1 and T2. At T3, using reputation with decay does achieve significantly better performance than using reputation without decay, but neither system significantly outperforms the system that does not use reputation. The primary effect of reputation decay is to induce mobility in stagnant agents, but mobility becomes less important with a higher agent-to-node ratio.

These results suggest that the best factor-level combination for using reputation with decay involves three agents and a time of classification of T2. Furthermore, mov-

Table 9. Four Agent Classification Accuracy Over 30 Runs

System	Class. Time	Min	Mean	Max	Std Dev
No Rep.	T1	0.37	0.56	<i>0.77</i>	0.10
	T2	0.44	0.67	0.80	0.09
	T3	<i>0.57</i>	0.67	<i>0.80</i>	0.06
	T4	0.44	0.62	0.74	0.08
Rep w/out Decay	T1	0.40	<i>0.61</i>	0.74	0.08
	T2	0.55	0.73	0.84	0.07
	T3	0.47	0.66	0.76	0.07
	T4	<i>0.51</i>	<i>0.63</i>	<i>0.78</i>	0.07
Rep w/ Decay	T1	<i>0.45</i>	0.60	0.73	<i>0.06</i>
	T2	0.60	0.71	0.82	<i>0.06</i>
	T3	0.52	<i>0.69</i>	0.78	<i>0.06</i>
	T4	0.50	0.61	0.73	0.06

Table 10. P Values For Two-Sided Wilcoxon Rank Sum Test for Four Agent System

	Median			P-Value		
	A	B	C	A vs B	A vs C	B vs C
T1	0.570	0.595	0.595	0.048	0.151	0.441
T2	0.690	0.715	0.720	0.019	0.064	0.420
T3	0.670	0.670	0.700	0.482	0.160	0.025
T4	0.615	0.635	0.615	0.756	0.407	0.139

ing from the three agent case to the four agent case reveals the declining importance of reputation decay as the number of agents in the system is increased.

While reputation with decay is not a panacea, it is clearly beneficial under specific conditions and never harmful under any of the tested conditions. Where does the benefit come from? Observe that the only way for agents to achieve stability in a system using reputation with decay is to find, collectively, nodes that each contribute a unique set of information capable of swaying combined classification determinations, but only such that the majority of combined classifications agree with each other.

```

MASON Version 14. For further options, try adding '-help' at end.
DDoS scenario targeting [1]:
- attackers at nodes [36] [4] [6] [15] [9] [42]
- attack to commence at t = 50
Job: 0 Seed: 1298872018140
Starting mfire.network.implementations.DDoSTopgenNetwork
loading topgen01.topgen
t = 1; pinging [19 / 17] -> [1]:
t = 17; Reply from [1]: time=16
t = 18; Reply from [1]: time=16
t = 19; Reply from [1]: time=16
t = 20; Reply from [1]: time=16
t = 21; Reply from [1]: time=16
t = 22; Reply from [1]: time=16
t = 23; Reply from [1]: time=16
t = 24; Reply from [1]: time=16
t = 25; Reply from [1]: time=16
t = 26; Reply from [1]: time=16
[19] Ping statistics for [1]:
    Packets: Sent = 10, Received = 10, Lost = 0 (0% loss).
Round trip times in simulation steps:
    Minimum = 16, Maximum = 16, Average = 16.0
t = 50; DoS attack launched: [9] -> [1]
t = 50; DoS attack launched: [4] -> [1]
t = 50; DoS attack launched: [36] -> [1]
t = 50; DoS attack launched: [6] -> [1]
t = 50; DoS attack launched: [42] -> [1]
t = 50; DoS attack launched: [15] -> [1]
t = 100; pinging [19 / 30] -> [1]:
t = 131; Reply from [1]: time=26
t = 131; Reply from [1]: time=23
t = 133; Reply from [1]: time=26
t = 139; Reply from [1]: time=33
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
[19] Ping statistics for [1]:
    Packets: Sent = 10, Received = 4, Lost = 6 (60% loss).
Round trip times in simulation steps:
    Minimum = 23, Maximum = 33, Average = 27.0

```

Figure 16. MFIRE DDoS Attack Test: Console Output

Any other case indicates that some of the nodes provide either redundant, or worse, misleading information.

5.2.2 MFIRE and Associated Network Simulation Environment Assessment: Results.

This section presents the results of the qualitative tests for MFIRE and its associated network simulation environment. These include tests of DDoS, scan, and worm attacks, as well as tests validating MFIRE’s communications and flow of execution.

5.2.2.1 MFIRE: DDoS Test.

This test creates a DDoS attack scenario involving multiple DoS attackers, a single target, and a ping process that pings the target before and after the attack starts. From the console output in Figure 16, the ping process sends two sets of pings, with

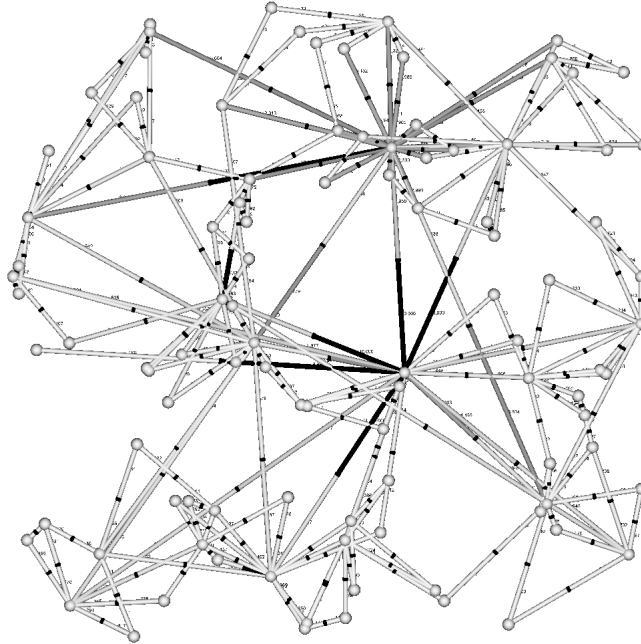


Figure 17. MFIRE DDoS Attack Test: Visualization

10 pings in each set. The ping process sends one ping on each timestep and receives 100% response for the first set of pings with no variance in delay. The DDoS then commences from six sources. A short time later, the ping process starts sending the second set of pings. Responses are received for the first four. The time delay of these responses is significantly longer than what was experienced for the previous set of pings, with the last taking more than twice as long to return (33 timesteps vs. 16). The cause of this delay is the random rerouting of the packets when the preferred link is full (see Section 3.4.4). The rest of the pings fail, for a total loss rate of 60%.

Figure 17 provides visual indication of what happens in this scenario. In this visualization, each link is divided in half by a black band. For a given node with an attached link, the half of the link directly connected to it shows inbound traffic flow, while the half on the other side of the band shows outbound traffic flow to the adjacent node. The shade of each half link changes according to load: white, on one end of the spectrum, represents zero load; black, on the other end, represents

```

[1] scanned nodes [18] - [19], ports 40 - 60

[18] profile:
    40    closed
    41    closed
    42    closed
    43    closed
    44    closed
    45    closed
    46    closed
    47    closed
    48    closed
    49    open
    50    closed
    51    closed
    52    closed
    53    closed
    54    closed
    55    closed
    56    closed
    57    closed
    58    closed
    59    closed
    60    closed

[19] profile:
    40    closed
    41    open/filtered
    42    closed
    43    closed
    44    closed
    45    closed
    46    closed
    47    closed
    48    closed
    49    closed
    50    closed
    51    closed
    52    closed
    53    closed
    54    open
    55    closed
    56    closed
    57    closed
    58    closed
    59    closed
    60    closed

Scan took 290 steps to complete.

```

Figure 18. MFIRE: Figure 9 redisplayed to show block scan conducted by the Scan-Process class

maximum load.

Observe that most of the links inbound to the target are at or near maximum capacity. Spillover effects are evident at the adjacent node toward the top of the figure. This qualitative test demonstrates the desired effect: reliable ping response in the absence of a DDoS attack, and unreliable response in its presence.

5.2.2.2 MFIRE: Scan Test.

Figure 18 demonstrates successful operation of the scan, in which the report shows the difference in the responses to the scan packets. This Figure shows evidence of two

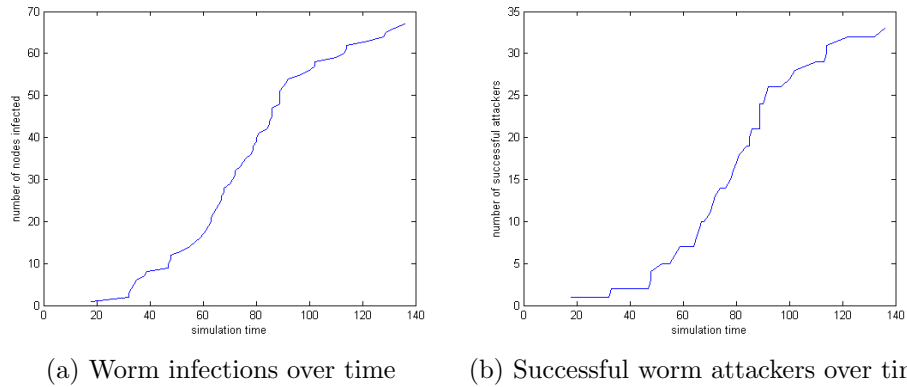


Figure 19. Worm attack growth charts

InsecureProcesses, one on each node, having replied to the scan packet with “OK.” It furthermore shows evidence of one unknown process which returned no response to the scan packet. For all other ports, the node sent “UNREACHABLE” back to the scanner. The desired effect is achieved and the test is successful.

5.2.2.3 MFIRE: Worm Test.

The Worm test creates a worm that makes one propagation attempt (attack) each timestep. The environment has four vulnerabilities. Each node has a resident InsecureProcess with at least one and up to four of these vulnerabilities. Each vulnerability is assigned a per-attack success rate (assuming a matching exploit) of 20%. The worm is armed with exploits for two of the four vulnerabilities, selected randomly.

Figure 19 shows the worm growth rates. Figure 19a shows the number of infected nodes over time, while Figure 19b shows the number of infected nodes having at least one successful attack over time. In both cases, exponential growth is evident for the majority of the simulation, until growth tapers off because most of the hosts vulnerable to one of the worm’s exploits have already been infected.

Figure 20 provides a visualization of the infected network. In this scenario, the scale is set to ‘regional’, allowing link lengths up to 10 units. Black bands separate

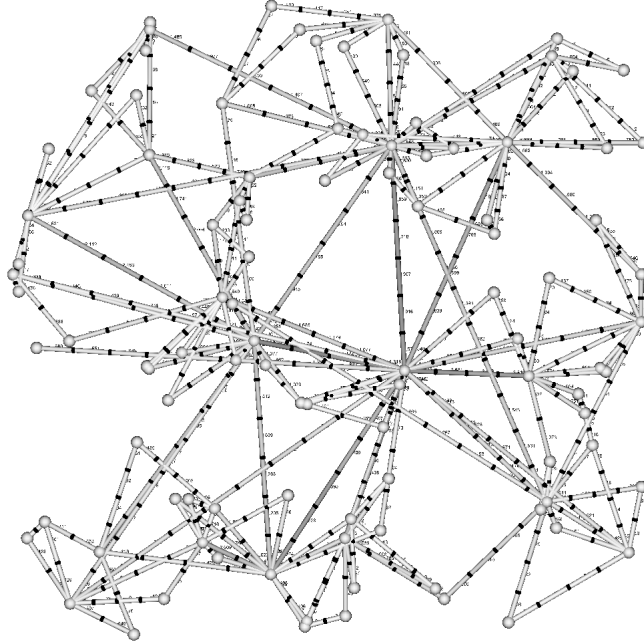


Figure 20. Worm attack visualization

each of the segments of the link, but one black band divides the link in half. Each half of the link is visually representative of the same concept as used in the DDoS test case: all traffic on each half of the link flows toward the node to which it is directly attached, thus providing a visual separation of the full duplex nature of the link.

While no single link in Figure 20 appears saturated as is the case in the DDoS scenario, many links are sustaining high loads at this point in the simulation, which is at approximately $t = 120$ or near the point at which the growth of the worm slows for lack of uninfected and vulnerable targets.

Having demonstrated the desired qualitative effects, this worm attack test is successful. As the third of three complex behaviors, the successful implementation of the worm attack demonstrates the ability of MFIRE's network simulation to support a range of flow-based attacks, satisfying two research objectives:

- *Develop an effective network simulation environment appropriate for the problem scope.*

- *Validate the proper functioning of simulated malicious traffic.*

5.2.2.4 MFIRE Communications and Execution Flow Tests.

The remaining tests involve the communications protocol and execution flow of MFIRE. Eleven test cases demonstrating a range of communications failures are implemented. The details of these test cases are provided in Appendix C. All test cases are currently successful.

5.3 Summary

This chapter presents the results of classification performance evaluation for MASNAC, our first design iteration of a multi agent network activity classifier, as well as qualitative system validation for the second design iteration, MFIRE: the network simulation environment and malicious traffic scenarios as well as the multi agent system.

Quantitative testing of MASNAC performance and qualitative validation of MFIRE operation demonstrates achievement of the research objectives:

1. *Develop an effective network simulation environment appropriate for the problem scope.*
2. *Validate the proper functioning of simulated malicious traffic.*
3. *Validate the proper command, control, and communications in the multi agent intrusion detection system.*
4. *Study the effects of several factors on classification accuracy.* Factors include the number of agents, the time in the scenario at which classification occurs, and the use of reputation (i.e. not used, used without decay, and used with decay).

The next chapter contains conclusions and highlights opportunities for future research.

VI. Conclusions and Future Research

This chapter highlights the successes and opportunities resulting from our research investigation concerning the development of a multi agent system for flow-based intrusion detection. Section 6.1 discusses some observations of the first and second design iterations, that is, of MASNAC and MFIRE respectively. Section 6.2 highlights opportunities for future research activity. Finally an overall research summary is presented.

6.1 Conclusions

The critical need for research investigations into distributed, flow-based intrusion detection systems is stated in Chapter I. As noted by Sperotto et al. [139], “distributed detection is particularly important... because the amount of traffic on high-speed networks is still increasing, suggesting that scalability will remain an issue in the future.”

Our successful research effort develops, in two iterations (see Chapters III and IV), a unique multi agent system designed to engage in flow-based intrusion detection in a distributed way. The major innovations of this effort include: 1) the use of reputation as a means of influencing the mobility patterns of the agents; and 2) a network simulation environment for MASON with the associated, animated visualization.

Both MAS classifiers involved the use of a classification technique at the local, Agent level (minimum euclidean distance for MASNAC / Support Vector Machine for MFIRE), the collection of local classification decisions at a central point, and the use of the majority vote as the MAS classification output.

Our research hypothesis from Chapter I is validated, indicating that we can increase the effectiveness of a flow-based, multi agent network attack classifier by:

1. Employing reputation to motivate agents to move when perceived as not providing useful information to peers;
2. Decaying the reputation to provide further impetus for agents to find the best vantage points.

with qualification; the benefit is observed for specific conditions. But, under no tested condition did reputation with or without decay perform worse than not using decay. See Section 5.2.1.

The testing and analysis conducted in Chapter V indicate that the Chapter I objectives are achieved, and thus, the hypothesis should be accepted for the initial model.

Recall that self-organization is the ability of a system to adapt in pursuit of better performance (see Section 2.5). The multi agent system reconfigures itself (in terms of spatial distribution of the agents) through the use of reputation. Because performance gains in certain conditions are attributable to the use of reputation, the multi agent system exhibits evidence of self-organization.

In the second design iteration, MFIRE, the desired functionality of the network simulation environment and the malicious activity classes has been achieved, demonstrated in the validation tests discussed in Section 5.2.2. The basic operation of the multi agent system in terms of flow of execution and communication is also validated.

6.2 Future Research Activity

The most immediate need for future research is a total, comprehensive performance evaluation of MFIRE. Factorial design can investigate the effects of various factors individually and jointly for increased understanding of how to set system parameters for effective performance. The objective is to determine the effectiveness

of the support vector machine for classification in this domain, as well as the effectiveness of a MOEA-produced rating table, compared with the effectiveness of our human-designed rating table.

One of the products that evolved out of our research is a set of network simulation model refinements and extensions that, if pursued, may increase the range and depth of MFIRE impact. These include:

- Reinstating the node failure effect used in the first iteration (see Section 4.2.1), with a closer fit to real-world empirical observations;
- Modeling asynchronous communications in finer detail, including finer-grained random delays and noise on the line corrupting messages;
- Exploring alternative topology generators, such as RealNet [37], [36], and comparing with current topology generation to evaluate impact on system performance;
- Implementing nodes that handle routing more intelligently when faced with network congestion;
- More complex attacks, including botnets and attacks that first conduct reconnaissance via scans.

Finally, there are several key areas for future research investigations to improve the multi agent classifier. They include:

- Using an existing agent framework such as JACK, Cougaar, or JADE to leverage tools for agent communications and interoperability [105];
- Feature generation to more fully investigate the feature space and thereby find such features as will increase the efficiency and effectiveness of classification;

- Implement a distributed reputation system and compare with the existing, centralized approach;
- Evaluate different classifiers and compare performance, including mixtures of different types of classifiers across the agent population;
- Evaluation of different MOEAs for best general results in producing rating tables that lead to effective MAS behaviors.

6.3 Overall Summary

This research effort validated our qualified hypothesis that reputation, when used to influence agents to seek better vantage points on the network, generally leads to improvements in the multi agent system's network attack classification performance for specific conditions. This is an important contribution that supports our goal of developing an effective, flow-based, multi agent system for inter-AS network attack classification.

The field of research for distributed, flow-based intrusion detection is new and wide open for exploration. In the fight to protect the inestimable advantages of our networks of autonomous systems, whether for civilian applications or in support of operations conducted by the United States military with which we are affiliated, one cannot afford to overlook multi-agent, flow-based intrusion detection techniques. It is our hope that the proof-of-concept simulation software implemented in this research be further investigated, or serve as an inspiration for parallel efforts, with the possibility of creating an effective and scalable complement to a suite of network defense capabilities.

Appendix A. Evolutionary Algorithms: Details and Applications

This appendix provides formalisms for generic, single objective evolutionary algorithms as well as some examples of how evolutionary algorithms have been applied.

1.1 Evolutionary Algorithms: Details

For a single objective evolutionary algorithm, Bäck defines an EA as an 8-tuple[8]:

$$\text{EA} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$$

where I is the space of individuals (analagous to the feasible region M). $\Phi : I \rightarrow \mathbb{R}$ is a fitness function that assigns real values to individuals based on performance with respect to the *objective*.

$$\Omega = \{\omega_{\Theta_1}, \dots, \omega_{\Theta_z} | \omega_{\Theta_i} : I^\lambda \rightarrow I^\lambda\} \cup \{\omega_{\Theta_0} : I^\mu \rightarrow I^\lambda\}$$

is a set of probabilistic genetic operators ω_{Θ_i} such as mutation or recombination. Each operator is controlled by specific parameters summarized in the sets $\Theta_i \subset \mathbb{R}$.

The *selection operator* is

$$s_{\Theta_s} : (I^\lambda \cup I^{\mu+\lambda}) \rightarrow I^\mu$$

which may change the number of individuals from λ or $\lambda + \mu$ to μ , where $\mu, \lambda \in \mathbb{N}$ and $\mu = \lambda$ is permitted. Here, λ is the number of offspring and μ is the number of parents in the population. As with the probabilistic genetic operators, the selection operator may make use of a set of parameters; specifically, Θ_s .

The *termination criterion* for the EA is expressed by $\iota : I^\mu \rightarrow \{\text{true}, \text{false}\}$. The

generation transition function $\Psi : I^\mu \rightarrow I^\mu$ specifies the complete process by which we transition from the population P of the current generation to that of the subsequent generation:

$$\Psi = s \circ \omega_{\Theta_{i_1}} \circ \dots \circ \omega_{\Theta_{i_j}} \circ \omega_{\Theta_{i_0}}$$

$$\Psi(P) = s_{\Theta_s}(Q \cup \omega_{\Theta_{i_1}}(\dots(\omega_{\Theta_{i_j}}(\omega_{\Theta_{i_0}}(P))))\dots))$$

Here, $\{i_1, \dots, i_j\} \subseteq \{1, \dots, z\}$. That is to say, Ψ is permitted to use a subset of available, parameterized operators. Finally, $Q \in \{\emptyset, P\}$. This way, Ψ may specify whether selection includes the population as it existed prior to the current generation's transformation along with the transformed population. The operator $\omega_0 : I^\mu \rightarrow I^\lambda$ serves to change the population size so that the required λ offspring individuals are produced from the μ parents. Selection reverts the population size to μ .

A *population sequence* $P(t+1) = \Psi(P(t)), \forall t \geq 0$ naturally results from Ψ , where t denotes the generation. Members of $P(0)$ are typically initialized randomly, but $P(0)$ may also be generated from a specified starting point.

Genetic operators come in many varieties but are broadly categorized by the number of “parents” involved. Mutation is an example of an asexual genetic operator, while recombination is typically sexual, involving two parents, but could be extended (without basis in biology) to involve arbitrarily many parents.

Permissible mutations depend on the representation of the individual. A common case is the bitstring representation. A bit-flip mutation flips the selected bit with some probability. Similar single-parameter mutations for parameters in other domains may involve an attempt to constrain the mutation so that the new value is “close” by some measure to the old value (for example via shifting by a value drawn from a gaussian distribution).

Other genetic operators are similarly representation-dependent. Common recombination operators include one-point crossover, where a parameter position is selected at random via some distribution (e.g. uniform), and the offspring is formed by copying the all parameter values in the first parent up to and including the crossover point, and taking the rest of its values from the second parent after the crossover point. In general, n -point crossover extends this concept to n randomly selected crossover points.

Algorithm 4 Outline of an Evolutionary Algorithm

```

t := 0
initialize  $P(0) := \{\vec{a}_1(0), \dots, \vec{a}_\mu(0)\} \in I^\mu$ 
evaluate  $P(0) : \{\Phi(\vec{a}_1(0)), \dots, \Phi(\vec{a}_\mu(0))\}$ 
while  $\iota(P(t)) \neq \mathbf{true}$  do
  recombine:  $P'(t) := r_{\Theta_r}(P(t))$ 
  mutate:  $P''(t) := m_{\Theta_m}(P'(t))$ 
  evaluate:  $P''(t) : \{\Phi(\vec{a}'_1(t)), \dots, \Phi(\vec{a}'_\lambda(t))\}$ 
  select:  $P(t+1) := s_{\Theta_s}(P''(t) \cup Q)$ 
   $t := t + 1$ 
end while

```

The general outline of an Evolutionary Algorithm is presented by Bäck in algorithm 4 [8].

1.1.1 Evolutionary Algorithms: Applications.

A particularly pertinent example of the application of Evolutionary Algorithms to pattern recognition comes from Radtke et al. [130]. The authors apply Multi-Objective Genetic Algorithms (MOGAs) to two parts of a handwritten character recognition system. First, they use the Multi Objective Memetic Algorithm (MOMA) [131] to extract a small set of effective features. The fitness function minimizes both the dimensionality of the feature set and the classification error rate of a projection distance (PD) classifier [86] on a validation set.

The PD is only used as a wrapper (see section 2.2.1.2) for evaluation of the extracted feature set. In the next step, these features are used to train a diverse set K of p classifiers to be used as candidates for the final Ensemble of Classifiers [87]. Again, a MOGA is applied, in this case the popular Non-Dominated Sorting Genetic Algorithm (NSGA-II) [42], for the subset selection from K . The fitness of each solution (a binary vector of length p indicating membership of each K_i in the candidate ensemble) in this case is based on the performance of the ensemble on a validation set.

Another pattern recognition example is presented in [162], in which the authors study feature selection using single and multi-objective memetic frameworks.

For examples of how these concepts have been applied to intrusion detection, see [10] for an analysis of MOGAs used to identify encrypted traffic. In [68], Haag applies a multi-objective artificial immune system to some public network traffic data sets to detect intrusions.

Appendix B. MFIRE System Details

This appendix provides some additional MFIRE details. In section 2.1, the messages used by MFIRE are detailed. Section 2.2 lists the fourteen observations collected by MFIRE agents for use in derived features.

2.1 MFIRE: Messages

The figures provided in this section show the messages used in MFIRE. In each figure, the left side is used for the sender. The type of the message is displayed first, and below it, the format. The format is essential for extracting message components from the packet's payload, which itself is a single string. On the right side of each figure, we show the actions that are taken by the recipient.

Figure 21 shows the messages sent from the controller and received by agents.

Figure 22 shows the messages sent from agents and received by the controller.

Figure 23 shows the SHARE message used for feature value exchange between agents.

Figure 24 shows the messages involved in agent migration. MIGRATE is sent by an agent that received MOVE from the controller previously. It is sent to the AgentManager at the migration destination node. The MIGRATE message contains all information required to reinstantiate the agent at the distant end. MIGRATEACK is sent by an AgentManager that received a MIGRATE message previously. It is sent to the AgentManager at the node where the original copy of the migrating agent still resides. The AgentManager that receives MIGRATEACK terminates the agent.

2.2 MFIRE: Observations

Each observation in MFIRE represents a traffic statistic collected over the duration of a single timestep. These are used to derive feature values.

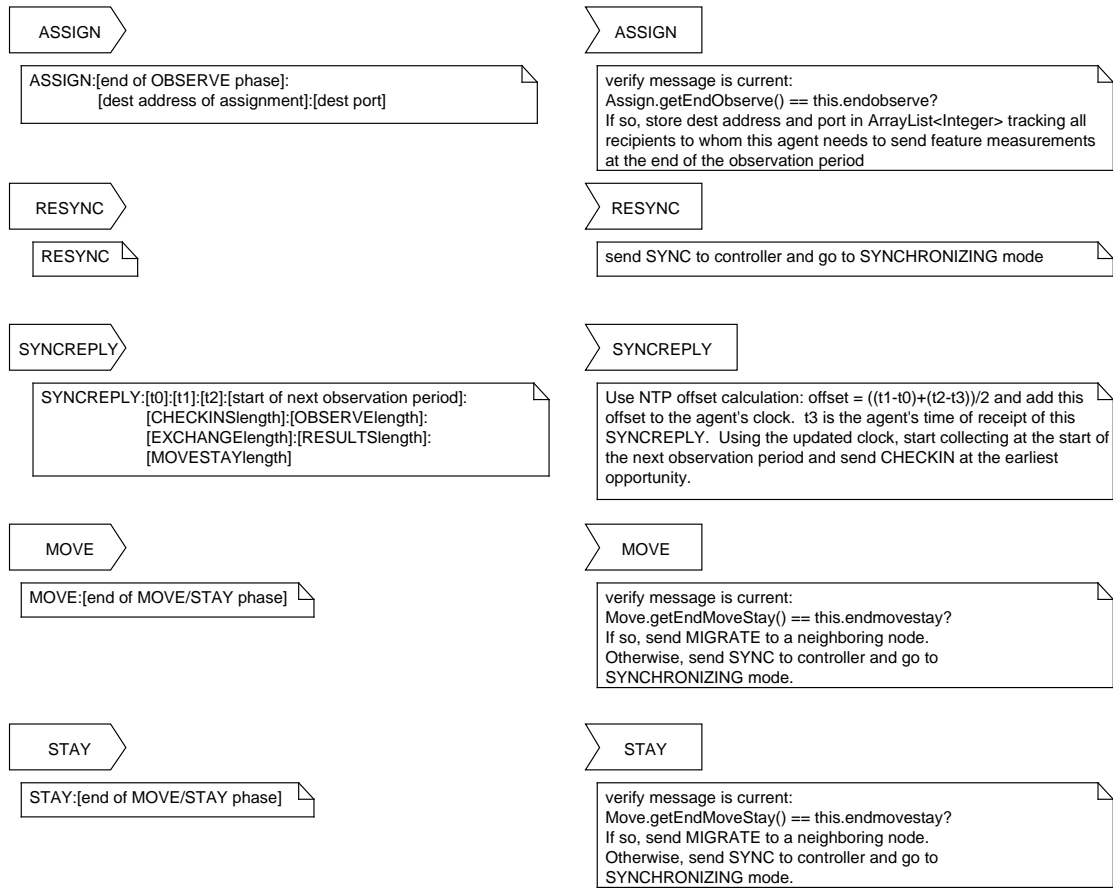


Figure 21. MFIRE: Messages sent by the controller and received by agents

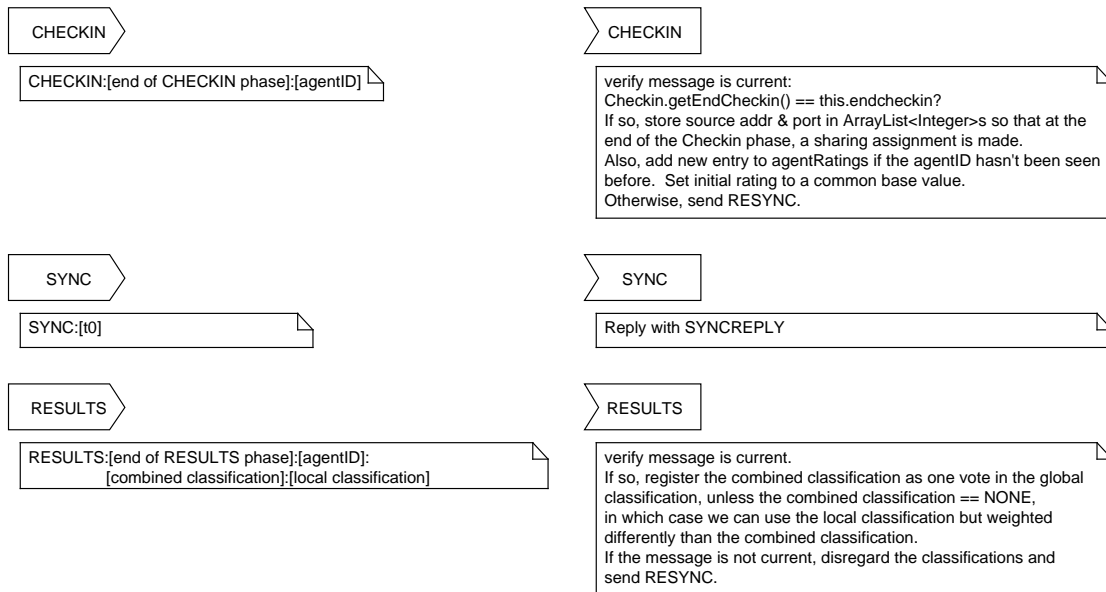


Figure 22. MFIRE: Messages sent by agents and received by the controller

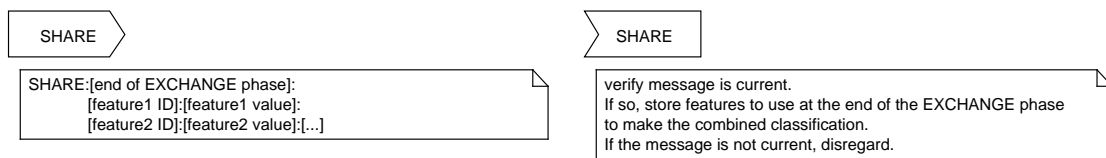


Figure 23. MFIRE: Messages sent by agents to other agents

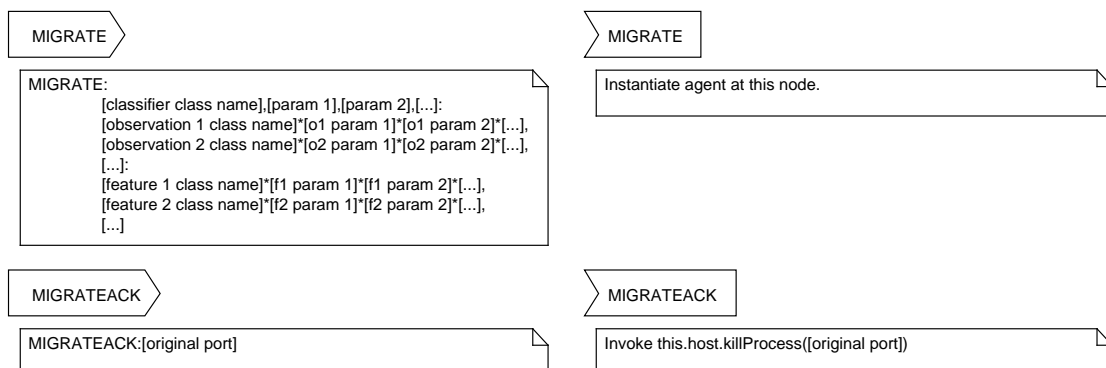


Figure 24. MFIRE: Messages involved in agent migration

The fourteen observations collected by agents in MFIRE:

1. Average number of bytes per $\langle destaddr, destport \rangle$ -tuple
2. Average number of bytes per $\langle sourceaddr, sourceport \rangle$ -tuple
3. Number of distinct destination addresses
4. Number of distinct $\langle destaddr, destport \rangle$ -tuples
5. Number of distinct destination ports
6. Ratio of destination ports to destination addresses
7. Total number of inbound bytes
8. Total number of inbound packets
9. Ratio of packets to $\langle destaddr, destport \rangle$ -tuples
10. Ratio of packets to $\langle sourceaddr, sourceport \rangle$ -tuples
11. Number of distinct source addresses
12. Number of distinct $\langle sourceaddr, sourceport \rangle$ -tuples
13. Number of distinct source ports
14. Ratio of source ports to source addresses

Clearly there are many linear dependencies in this set of observations. Care must be exercised when performing feature selection from this set.

Appendix C. Test Plan for MFIRE Command, Control, and Communications

Communication in a network environment is fraught with peril. Packet loss is incurred by noise on the line or disruptions in network services. Links are sometimes severed, routers and switches fail, and processes on either end of a traffic flow may shut down unexpectedly or be unacceptably delayed for a variety of reasons.

The challenge to network protocol design is to handle such failures gracefully.

MFIRE is designed to operate in the face of failure. The principle cause of communications failures is saturated links due to denial of service attacks and worm outbreaks. Nodes do attempt to route around the problem (in a very simple way), but when multiple inbound links to the destination are saturated, a packet's Time to Live (TTL) is quickly exhausted, and the packet is discarded.

The system includes eleven types of messages. There are two Controller states and six Agent states, resulting in twelve combined states. In each of these states, one of the eleven types of messages may or may not be expected by either the Agent or the Controller.

Suffice it to say there are three general categories of test cases to consider: 1) normal operation; the appropriate message type is received and is in-sync with what the receiving party expected; 2) message loss; the Agent or the Controller expected a message that never arrived; 3) unexpected message received; the receiving party either did not expect a message of that type for the current state, or the message is out-of-sync with the receiver.

Determining whether a message is in-sync or out-of-sync is made possible by requiring each message to include a time-based field, typically indicating the sender's value for the end of the current state.

Exhaustively testing all combinations of messages (including losses and in- or out-

of-sync cases), Controller states, and Agent states requires well over one hundred tests. Many of these have considerable conceptual overlap; for example, if the Agent is in the AWAITINGENDOBSERVE state and receives anything but ASSIGN from the Controller (e.g. STAY, MOVE, RESYNC), the situation is the same: receipt of an unexpected message.

The test cases listed below focus on the most important aspects of ensuring the multi agent system has a robust communications protocol.

Test Case	Expected Observable Behaviors
Synchronization - no failures	<ul style="list-style-type: none"> • Agents send SYNC to Controller • Controller responds with SYNCREPLY • Agent uses SYNCREPLY payload to adjust its clock to match the Controller's • when packets are not delayed, the clocks for both Agent and Controller should have the same value at the conclusion of the Process phase and thereafter

Synchronization - Single SYNC Loss and Recovery	<ul style="list-style-type: none">• Agent's active status displayed with each call to processMessages()• Agent sends SYNC to Controller with minimum TTL• SYNC TTL expires before reaching Controller• Agent times out and sends SYNC again with normal TTL• Controller responds with SYNCREPLY• Agent uses SYNCREPLY payload to adjust its clock to match the Controller's• when packets are not delayed, the clocks for both Agent and Controller should have the same value at the conclusion of the Process phase and thereafter
-------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p>Synchronization - Multiple SYNC Losses and Agent Deactivation</p>	<ul style="list-style-type: none">• Agent's active status displayed with each call to processMessages()• Agent sends SYNC to Controller with minimum TTL• SYNC TTL expires before reaching Controller• Agent times out and sends SYNC again• repeat until MAXTIMEOUTS exceeded• Agent deactivates and is removed from the schedule• Following deactivation, no more status displays from this agent
----------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p>Synchronization - Single SYNCREPLY Loss and Recovery</p>	<ul style="list-style-type: none"> • Agent sends SYNC to Controller • Controller responds with SYNCREPLY with minimum TTL • SYNCREPLY TTL expires before reaching Agent • Agent times out and sends SYNC again • Controller responds with SYNCREPLY • Agent uses SYNCREPLY payload to adjust its clock to match the Controller's • when packets are not delayed, the clocks for both Agent and Controller should have the same value at the conclusion of the Process phase and thereafter
<p>Synchronization - Multiple SYNCREPLY Losses and Agent Deactivation</p>	<ul style="list-style-type: none"> • Agent sends SYNC to Controller • Controller responds with SYNCREPLY with minimum TTL • SYNCREPLY TTL expires before reaching Agent • Agent times out and sends SYNC again • repeat until MAXTIMEOUTS exceeded • Agent deactivates and is removed from the schedule • Following deactivation, no more status displays from this agent

Agent Desync and Recovery	<ul style="list-style-type: none"> • Agent synchronizes with Controller • Agent sends CHECKIN to Controller • Agent receives SHARE from peer • Agent sends RESULT to Controller • Controller sends STAY to Agent • Agent clock artificially advanced 1000 ticks • Agent sends CHECKIN with out-of-sync value in the end-of-check-in field • Controller responds with RESYNC • Agent synchronizes per the “Synchronization - no failures” case • Agent sends CHECKIN at the next window of opportunity
Agent Fails to Receive SHARE	<ul style="list-style-type: none"> • Agent sends CHECKIN to Controller • Controller sends ASSIGN to peer for the checked-in Agent • Peer sends SHARE with minimum TTL • SHARE TTL expires before reaching Agent • Agent sends RESULT to Controller with NOCOMBINEDCLASS value in the “combined classification” field

<p>Agent Fails to Receive STAY / MOVE</p>	<ul style="list-style-type: none"> • Agent sends RESULT to Controller • Controller sends STAY with minimum TTL • STAY TTL expires before reaching Agent • Agent synchronizes per the “Synchronization - no failures” case • Agent sends CHECKIN at the next window of opportunity
<p>Controller Fails to Receive RESULT</p>	<ul style="list-style-type: none"> • Agent sends RESULT to Controller with minimum TTL • RESULT TTL expires before reaching Controller • Controller publishes result indicating number of votes cast = one less than the number of agents • Controller sends STAY (STAY or MOVE is sent to all agents that sent CHECKIN regardless of whether a RESULT was received) • Agent sends CHECKIN

<p>Agent Receives Unexpected Message</p>	<ul style="list-style-type: none"> • Agent sends RESULT • Agent artificially changes mode to AWAITINGEXCHANGE • Controller sends STAY • Agent synchronizes per the “Synchronization - no failures” case
<p>Controller Receives Unexpected Message</p>	<ul style="list-style-type: none"> • Agent sends CHECKIN • While in AWAITINGEXCHANGE, Agent sends CHECKIN again • Controller responds with RESYNC • Agent synchronizes per the “Synchronization - no failures” case

Bibliography

- [1] “ATLAS: Global Network Threat Analysis”, 2011. URL <http://www.arbornetworks.com/en/atlas-global-network-threat-analysis.html>.
- [2] “Port Numbers”, February 2011. URL <http://www.iana.org/assignments/port-numbers>.
- [3] Alaya, I., C. Solnon, and K. Ghedira. “Ant colony optimization for multi-objective optimization problems”. *19th IEEE International Conference on Tools with Artificial Intelligence, 2007. ICTAI 2007*, volume 1. 2007.
- [4] Albert, R., H. Jeong, and A.L. Barabási. “Error and attack tolerance of complex networks”. *Arxiv preprint cond-mat/0008064*, 2000.
- [5] Amaldi, E. and V. Kann. “On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems”. *Theoretical Computer Science*, 209(1-2):237–260, 1998. ISSN 0304-3975.
- [6] Anderson, R., R. Böhme, R. Clayton, and T. Moore. “Security economics and european policy”. *Managing Information Risk and the Economics of Security*, 55–80, 2009.
- [7] Axelsson, S. “Intrusion detection systems: A survey and taxonomy”. 2000.
- [8] Bäck, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.
- [9] Bäck, T., F. Hoffmeister, and H.P. Schwefel. “A survey of evolution strategies”. R. Belew (editor), *Proceedings of the Fourth International Conference on Genetic Algorithms*, 2–9. Morgan Kaufmann, 1991.
- [10] Bacquet, C., A. Zincir-Heywood, and M. Heywood. “An Investigation of Multi-objective Genetic Algorithms for Encrypted Traffic Identification”. *Computational Intelligence in Security for Information Systems*, 93–100, 2009.
- [11] Bagrodia, R., R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. “Parsec: A parallel simulation environment for complex systems”. *Computer*, 31(10):77–85, 2002. ISSN 0018-9162.
- [12] Banks, J., B.L. Nelson, and D.M. Nicol. *Discrete-event system simulation*. Prentice Hall, 2009. ISBN 0136062121.
- [13] Bar, S., M. Gonen, and A. Wool. “A geographic directed preferential Internet topology model”. *Computer Networks*, 51(14):4174–4188, 2007. ISSN 1389-1286.

- [14] Barabási, A.L. and R. Albert. “Emergence of scaling in random networks”. *Science*, 286(5439):509, 1999.
- [15] Barford, P. and M. Crovella. “Generating representative web workloads for network and server performance evaluation”. *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 160. ACM, 1998. ISBN 0897919823.
- [16] Barr, R.S., B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. “Designing and reporting on computational experiments with heuristic methods”. *Journal of Heuristics*, 1(1):9–32, 1995. ISSN 1381-1231.
- [17] Baufreton, P. “SACRES: A step ahead in the development of critical avionics applications”. *Lecture notes in computer science*, 1999. ISSN 0302-9743.
- [18] Baufreton, P. “Visual notations based on synchronous languages for dynamic validation of gals systems”. *CCCT04 Computing, Communications and Control Technologies*, 2004.
- [19] Becchi, M. “From Poisson Processes to Self-Similarity: a Survey of Network Traffic Models”. 2008.
- [20] Bellifemine, F., A. Poggi, and G. Rimassa. “JADE—A FIPA-compliant agent framework”. *Proceedings of PAAM*, volume 99, 97–108. Citeseer, 1999.
- [21] Bellifemine, F., A. Poggi, and G. Rimassa. “Developing multi-agent systems with a FIPA-compliant agent framework”. *Software: Practice and Experience*, 31(2):103–128, 2001. ISSN 1097-024X.
- [22] Bellman, R. “Adaptive control processes: a guided tour”. *Princeton University Press*, 1:2, 1961.
- [23] Bhattacharyya, A. “On a measure of divergence between two statistical populations defined by probability distributions”. *Bull. Calcutta Math. Soc*, 35:99–109, 1943.
- [24] Bishop, C.M. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.
- [25] Bleuler, S., M. Laumanns, L. Thiele, and E. Zitzler. “PISA: a platform and programming language independent interface for search algorithms”. *Proceedings of the 2nd international conference on Evolutionary multi-criterion optimization*, 494–508. Springer-Verlag, 2003. ISBN 3540018697.
- [26] Bollobás, B. *Random graphs*. Cambridge University Press, 2001.

- [27] Brauckhoff, D., B. Tellenbach, A. Wagner, M. May, and A. Lakhina. “Impact of packet sampling on anomaly detection metrics”. *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 159–164. ACM, 2006. ISBN 1595935614.
- [28] Brenner, J. “Privacy and Security: Why Isn’t Cyberspace More Secure?” *Commun. ACM*, 53(11):33–35, 2010. ISSN 0001-0782.
- [29] Bu, T. and D. Towsley. “On distinguishing between Internet power law topology generators”. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, 638–647. IEEE, 2002. ISBN 0780374762. ISSN 0743-166X.
- [30] Calvert, K.I., M.B. Doar, and E.W. Zegura. “Modeling Internet topology”. *Communications Magazine, IEEE*, 35(6):160–163, June 1997. ISSN 0163-6804.
- [31] Chang, Chih-Chung and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [32] Chang, D.F., R. Govindan, and J. Heidemann. “An empirical study of router response to large BGP routing table load”. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 203–208. ACM, 2002. ISBN 158113603X.
- [33] Chang, R.K.C. “Defending against flooding-based distributed denial-of-service attacks: A tutorial”. *Communications Magazine, IEEE*, 40(10):42–51, 2002. ISSN 0163-6804.
- [34] Chapelle, O., P. Haffner, and V.N. Vapnik. “Support vector machines for histogram-based image classification”. *Neural Networks, IEEE Transactions on*, 10(5):1055–1064, 2002. ISSN 1045-9227.
- [35] Chen, Z., L. Gao, and K. Kwiat. “Modeling the spread of active worms”. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE Societies*, volume 3, 1890–1900. IEEE, 2003. ISBN 0780377524. ISSN 0743-166X.
- [36] Cheng, L. *Simulation and topology generation for large-scale distributed systems*. Ph.D. thesis, UNIVERSITY OF BRITISH COLUMBIA, 2009.
- [37] Cheng, L., N.C. Hutchinson, and M.R. Ito. “RealNet: A topology generator based on real internet topology”. *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, 526–532. IEEE, 2008.

- [38] Coello, C.A.C. and N.C. Cortés. “Solving multiobjective optimization problems using an artificial immune system”. *Genetic Programming and Evolvable Machines*, 6(2):163–190, 2005.
- [39] Coello, C.A.C., G.B. Lamont, and D.A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer-Verlag New York Inc, 2007.
- [40] Crucitti, Paolo, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. “Error and attack tolerance of complex networks”. *Physica A: Statistical Mechanics and its Applications*, 340(1-3):388 – 394, 2004. ISSN 0378-4371.
- [41] De Wolf, T. and T. Holvoet. “Emergence versus self-organisation: Different concepts but promising when combined”. *Engineering Self-Organising Systems*, 1–15, 2005.
- [42] Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan. “A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II”. *Parallel Problem Solving from Nature PPSN VI*, 849–858. Springer, 2000.
- [43] Dempster, M.B.L. *A Self-Organizing Systems Perspective on Planning for Sustainability*. Master’s thesis, University of Waterloo, School of Urban and Regional Planning, 1998.
- [44] Devroye, Luc. “Random variate generation in one line of code”. *Proceedings of the 28th conference on Winter simulation, WSC ’96*, 265–272. IEEE Computer Society, Washington, DC, USA, 1996. ISBN 0-7803-3383-7. URL <http://dx.doi.org/10.1145/256562.256623>.
- [45] Doar, M.B. “A Better Model for Generating Test Networks”. *Conference record*, 86. Institute of Electrical and Electronics Engineers, 1996.
- [46] Duda, Richard O., Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000. ISBN 0471056693.
- [47] Durillo, Juan J., Antonio J. Nebro, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. *jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics*. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, December 2006.
- [48] Eichen, M.W. and J.A. Rochlis. “With microscope and tweezers: An analysis of the internet virus of november 1988”. 1989. ISSN 1540-7993.
- [49] Erdos, P. and A. Rényi. “On the evolution of random graphs”. *Publications of the Mathematical Institute of the Hungarian Academy of Science*, 5:17–61, 1960.

- [50] Fabrikant, A., E. Koutsoupias, and C. Papadimitriou. “Heuristically optimized trade-offs: A new paradigm for power laws in the Internet”. *Automata, Languages and Programming*, 781–781, 2002.
- [51] Falliere, N., L.O. Murchu, and E. Chien. *W32. Stuxnet Dossier*. Technical report, Symantec, 2011.
- [52] Faloutsos, M., P. Faloutsos, and C. Faloutsos. “On power-law relationships of the internet topology”. *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 251–262. ACM, 1999. ISBN 1581131356.
- [53] Floyd, R.W. “Algorithm 97: shortest path”. *Communications of the ACM*, 5(6):345, 1962. ISSN 0001-0782.
- [54] Fosnock, C. “Computer Worms: Past, Present, and Future”. *East Carolina University*, 2005.
- [55] Franklin, S. and A. Graesser. “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”. *Intelligent Agents III Agent Theories, Architectures, and Languages*, 21–35, 1997.
- [56] Friedman, M. and A. Kandel. *Introduction to pattern recognition: statistical, structural, neural, and fuzzy logic approaches*. World Scientific Pub Co Inc, 1999. ISBN 9810233124.
- [57] Frost, V.S. and B. Melamed. “Traffic Modeling For Telecommunications Networks”. *IEEE Communications Magazine*, 1994.
- [58] Fujimoto, Richard M. “Parallel discrete event simulation”. *Commun. ACM*, 33:30–53, October 1990. ISSN 0001-0782. URL <http://doi.acm.org/10.1145/84537.84545>.
- [59] Gao, Yan, Zhichun Li, and Yan Chen. “A DoS Resilient Flow-level Intrusion Detection Approach for High-speed Networks”. 39 – 39. 2006. ISSN 1063-6927.
- [60] Gates, Carrie, Joshua J. McNutt, Joseph B. Kadane, and Marc I. Kellner. “Scan Detection on Very Large Networks Using Logistic Regression Modeling”. *Computers and Communications, IEEE Symposium on*, 0:402–408, 2006. ISSN 1530-1346.
- [61] Gertz, B. “2008 intrusion of networks spurred combined units”. The Washington Times, June 3 2010. URL <http://www.washingtontimes.com/new/2010/jun/3/2008-intrusion-of-networks-spurred-combined-units/>.
- [62] Gödel, K. *On formally undecidable propositions of Principia Mathematica and related systems*. Dover Pubns, 1992. ISBN 0486669807.

- [63] Gordon, J. “Pareto process as a model of self-similar packet traffic”. *Global Telecommunications Conference, 1995. GLOBECOM’95., IEEE*, volume 3, 2232–2236. IEEE, 2002. ISBN 0780325095.
- [64] Gotoh, T., K. Imamura, and A. Kaneko. “Improvement of NTP time offset under the asymmetric network with double packets method”. *Conference on Precision Electromagnetic Measurements, 2002, Conference Digest*. 2002.
- [65] Greengard, S. “The new face of war”. *Commun. ACM*, 53(12):20–22, 2010. ISSN 0001-0782.
- [66] Guyon, I. and A. Elisseeff. “An introduction to variable and feature selection”. *The Journal of Machine Learning Research*, 3:1157–1182, 2003. ISSN 1532-4435.
- [67] Guyon, Isabelle. “A Scaling Law for the Validation-Set Training-Set Size Ratio”. *AT & T Bell Laboratories*. 1997.
- [68] Haag, C.R., G.B. Lamont, P.D. Williams, and G.L. Peterson. “An artificial immune system-inspired multiobjective evolutionary algorithm with application to the detection of distributed computer network intrusions”. *Proceedings of the 6th international conference on Artificial immune systems*, 420–435. Springer-Verlag, 2007.
- [69] Halbwachs, N. and L. Mandel. “Simulation and verification of asynchronous systems by means of a synchronous model”. 2006. ISSN 1550-4808.
- [70] Hall, M.A. “Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning”. *Proceedings of the Seventeenth International Conference on Machine Learning*, 359–366. Morgan Kaufmann Publishers Inc., 2000. ISBN 1558607072.
- [71] Halley, J. et al. “Classification of emergence and its relation to self-organization”. *Complexity*, 13(5):10–15, 2008. ISSN 1099-0526.
- [72] Hancock, D. and G. Lamont. “Multi Agent Systems on Military Networks”. *IEEE Symposium on Computational Intelligence in Cyber Security*. 2011.
- [73] Hancock, D. and G. Lamont. “Reputation in a multi agent system for flow-based network attack classification”. *IEEE Symposium on Intelligent Agents*. 2011.
- [74] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2nd ed. 2009. corr. 3rd printing edition, February 2009. ISBN 0387848576. URL <http://www-stat.stanford.edu/~tibs/ElemStatLearn/>.

- [75] Hildebrandt, D., L. Bischofs, and W. Hasselbring. “RealPeer—A Framework for Simulation-Based Development of Peer-to-Peer Systems”. *Parallel, Distributed and Network-Based Processing, 2007. PDP’07. 15th EUROMICRO International Conference on*, 490–497. IEEE, 2007. ISBN 0769527841. ISSN 1066-6192.
- [76] Hofstadter, D.R. *Gödel, Escher, Bach: an eternal golden braid*. Basic books New York, 1999. ISBN 0465026567.
- [77] Holland, J.H. “Adaptation in natural and artificial systems”. *Ann Arbor MI: University of Michigan Press*, 1975.
- [78] Holloway, E. *Self organized multi agent swarms (SOMAS) for network security control*. Master’s thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, 2009.
- [79] Howden, N., R. Rönquist, A. Hodgson, and A. Lucas. “JACK intelligent agents-summary of an agent infrastructure”. *5th International Conference on Autonomous Agents*. Citeseer, 2001.
- [80] Huynh, T.D., N.R. Jennings, and N.R. Shadbolt. “An integrated trust and reputation model for open multi-agent systems”. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006. ISSN 1387-2532.
- [81] Igel, Christian, Tobias Glasmachers, and Verena Heidrich-Meisner. “Shark”. *Journal of Machine Learning Research*, 9:993–996, 2008.
- [82] Jansen, W.A. “Intrusion detection with mobile agents”. *Computer Communications*, 25(15):1392–1401, 2002. ISSN 0140-3664.
- [83] Jaszkiwicz, A. “MOMHLib++: Multiple Objective MetaHeuristics Library in C++”, 2005. URL <http://home.gna.org/momh/>.
- [84] Kim, J., S. Radhakrishnan, and S.K. Dhall. “Measurement and analysis of worm propagation on Internet network topology”. *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, 495–500. IEEE, 2005. ISBN 0780388143. ISSN 1095-2055.
- [85] Kim, M.S., H.J. Kang, S.C. Hung, S.H. Chung, and J.W. Hong. “A flow-based method for abnormal network traffic detection”. *IEEE/IFIP Network Operations and Management Symposium*, 599–612. 2004.
- [86] Kimura, F., S. Inoue, T. Wakabayashi, S. Tsuruoka, and Y. Miyake. “Handwritten numeral recognition using autoassociative neural networks”. *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 1, 166–171. IEEE, 2002. ISBN 0818685123.

- [87] Kittler, J., M. Hatef, R.P.W. Duin, and J. Matas. “On combining classifiers”. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, 2002. ISSN 0162-8828.
- [88] Knuth, D.E. *The Stanford GraphBase: a platform for combinatorial computing*. AcM Press, 1993. ISBN 0201542757.
- [89] Kohavi, R. and G.H. John. “Wrappers for feature subset selection”. *Artificial intelligence*, 97(1-2):273–324, 1997. ISSN 0004-3702.
- [90] Kong, J., M. Mirza, J. Shu, C. Yoedhana, M. Gerla, and S. Lu. “Random flow network modeling and simulations for DDoS attack mitigation”. *Communications, 2003. ICC’03. IEEE International Conference on*, volume 1, 487–491. IEEE, 2003. ISBN 0780378024.
- [91] Kotsiantis, S. B. “Supervised Machine Learning: A Review of Classification Techniques”. *Informatica*, 31:249–268, 2007.
- [92] Krekel, B. “Capability of the People’s Republic of China to Conduct Cyber Warfare and Computer Network Exploitation”. 2009.
- [93] Krishnamurthy, B. and W. Willinger. “What are our standards for validation of measurement-based networking research?” *ACM SIGMETRICS Performance Evaluation Review*, 36(2):64–69, 2008. ISSN 0163-5999.
- [94] Kuipers, B.J. “Qualitative simulation: then and now”. *Artificial intelligence in perspective*, MIT Press, Cambridge, MA, 1994.
- [95] Kurose, J. and K. Ross. *Computer Networking: A top-down approach*. Pearson Addison-Wesley, fifth edition edition, 2009.
- [96] Kursawe, F. “A variant of evolution strategies for vector optimization”. *Parallel Problem Solving from Nature*, 193–197, 1991.
- [97] Kursawe, F. “Evolution strategies for vector optimization”. *Preliminary Proceedings of the 10th International Conference on Multiple Criteria Decision Making*, 187–193. Citeseer, 1992.
- [98] Lakhina, Anukool, Mark Crovella, and Christophe Diot. “Characterization of network-wide anomalies in traffic flows”. *IMC ’04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 201–206. ACM, New York, NY, USA, 2004. ISBN 1-58113-821-0.
- [99] Lakhina, Anukool, Mark Crovella, and Christophe Diot. “Diagnosing network-wide traffic anomalies”. *SIGCOMM Comput. Commun. Rev.*, 34(4):219–230, 2004. ISSN 0146-4833.

- [100] Lakhina, Anukool, Mark Crovella, and Christophe Diot. “Mining anomalies using traffic feature distributions”. *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, 217–228. ACM, New York, NY, USA, 2005. ISBN 1-59593-009-4.
- [101] Lal, T., O. Chapelle, J. Weston, and A. Elisseff. “Embedded methods”. *Feature Extraction*, 137–165, 2006.
- [102] Larman, C. *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2001. ISBN 0130925691.
- [103] Liefoghe, A., L. Jourdan, T. Legrand, J. Humeau, and E.G. Talbi. “ParadisEO-MOEO: A software framework for evolutionary multi-objective optimization”. *Advances in Multi-Objective Nature Inspired Computing*, 87–117, 2010.
- [104] Liljenstam, M., Y. Yuan, BJ Premore, and D. Nicol. “A mixed abstraction level simulation model of large-scale Internet worm infestations”. *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, 109–116. IEEE, 2003. ISBN 0769518400. ISSN 1526-7539.
- [105] López, S., A. Brintrup, D. McFarlane, and D. Dwyer. “Selecting a multi-agent system development tool for industrial applications: a case study of self-serving aircraft assets”. *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*, 400–405. IEEE. ISSN 2150-4938.
- [106] Luke, Sean, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. “MASON: A New Multi-Agent Simulation Toolkit”. *Proceedings of the 2004 Swarmfest Workshop*. 2004.
- [107] Lyon, G.F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009. ISBN 0979958717.
- [108] Magoni, D. “nem: A software for network topology analysis and modeling”. *miscots*, 0364. Published by the IEEE Computer Society, 2002.
- [109] Mai, J., C.N. Chuah, A. Sridharan, T. Ye, and H. Zang. “Is sampled data sufficient for anomaly detection?”. *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 165–176. ACM, 2006. ISBN 1595935614.
- [110] Mann, D.E. and S.M. Christey. *Common Vulnerabilities and Exposures*. Technical report, Technical report, The MITRE Corporation, 1999. <http://www.cve.mitre.org/docs/cerias.html>.

- [111] Mason, R.L., R.F. Gunst, and J.L. Hess. *Statistical design and analysis of experiments: with applications to engineering and science*. Wiley-Interscience, 2003. ISBN 0471372161.
- [112] McCanne, S., S. Floyd, and K. Fall. “ns2 (network simulator 2)”. *last accessed: February, 23, 2010*.
- [113] McDonald, C. “The cnet network simulator”. *University of Western Australia*, 2003.
- [114] McDonald, Chris. “The cnet network simulator”. URL <http://www.csse.uwa.edu.au/cnet/index.html>.
- [115] Mead, N.R. and T. Stehney. “Security quality requirements engineering (SQUARE) methodology”. *Proceedings of the 2005 workshop on Software engineering for secure systemsbuilding trustworthy applications*, 1–7. ACM, 2005. ISBN 1595931147.
- [116] Meadows, D. “Leverage points”. *Places to Intervene in a System.. Hartland, Vermont, USA: The Sustainability Institute*, 1999.
- [117] Medina, A., A. Lakhina, I. Matta, and J. Byers. “BRITE: An approach to universal topology generation”. *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, 346–353. IEEE, 2002. ISBN 0769513158.
- [118] Mills, D., J. Martin, J. Burbank, and W. Kasch. “RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification”. *Internet Engineering Task Force*, 2010.
- [119] Milner, R. “Calculi for synchrony and asynchrony”. *Theoretical computer science*, 25(3):267–310, 1983. ISSN 0304-3975.
- [120] Milner, R. and University of Edinburgh. Department of Computer Science. *On relating synchrony and asynchrony*. Department of Computer Science, University of Edinburgh, 1980.
- [121] Milton, J.S. and J.C. Arnold. *Introduction to probability and statistics: principles and applications for engineering and the computing sciences*. McGraw-Hill Higher Education, 2002. ISBN 007246836X.
- [122] Mirkovic, J. and P. Reiher. “A taxonomy of DDoS attack and DDoS defense mechanisms”. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004. ISSN 0146-4833.
- [123] Montgomery, D.C. *Design and analysis of experiments*. John Wiley & Sons Inc, 7th edition, 2008. ISBN 0470128664.

- [124] Munz, G. and G. Carle. “Real-time Analysis of Flow Data for Network Attack Detection”. 100–108. may. 2007.
- [125] Park, Hyungwook and Paul A. Fishwick. “A GPU-Based Application Framework Supporting Fast Discrete-Event Simulation”. *Simulation*, 86:613–628, October 2010. ISSN 0037-5497. URL <http://dx.doi.org/10.1177/0037549709340781>.
- [126] Postel, J. “RFC 793: Transmission control protocol”, 1981.
- [127] Postel, J.B. “User Datagram Protocol. RFC 768”, 1980.
- [128] Prokopenko, M., F. Boschetti, and A.J. Ryan. “An information-theoretic primer on complexity, self-organisation and emergence”. *Complexity*, 15(1):11–28, 2009. ISSN 1099-0526.
- [129] Quittek, J., T. Zseby, and B. Claise. *S. Zander,” Requirements for IP Flow Information Export (IPFIX)*. Technical report, RFC 3917, October 2004.
- [130] Radtke, Paulo V. W., Robert Sabourin, and Tony Wong. “Classification system optimization with multi-objective genetic algorithms”. Guy Lorette (editor), *Tenth International Workshop on Frontiers in Handwriting Recognition*. Université de Rennes 1, Suvisoft, La Baule (France), 10 2006. URL <http://hal.inria.fr/inria-00104200/en/>.
- [131] Radtke, P.V.W., T. Wong, and R. Sabourin. “A multi-objective memetic algorithm for intelligent feature extraction”. *Evolutionary Multi-Criterion Optimization*, 767–781. Springer, 2005.
- [132] Resnick, P. and R. Zeckhauser. “Trust among strangers in Internet transactions: Empirical analysis of eBay’s reputation system”. *Advances in Applied Microeconomics: A Research Annual*, 11:127–157, 2002. ISSN 0278-0984.
- [133] Russell, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2 edition, December 2002. ISBN 0137903952.
- [134] Say, AC and H.L. Akin. “Sound and complete qualitative simulation is impossible”. *Artificial Intelligence*, 149(2):251–266, 2003. ISSN 0004-3702.
- [135] Sellke, S.H., N.B. Shroff, and S. Bagchi. “Modeling and automated containment of worms”. *IEEE Transactions on Dependable and Secure Computing*, 71–86, 2007. ISSN 1545-5971.
- [136] Shoham, Y. and K. Leyton-Brown. *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. Cambridge Univ Pr, 2008. ISBN 0521899435.
- [137] Soergel, D. “jlibsvm”, 2009. URL <http://dev.davidsoergel.com/trac/jlibsvm>.

- [138] Spatharis, A., I. Foudalis, M. Gjoka, P. Krouska, C. Amanatidis, C. Papadimitriou, and M. Sideri. “Improved tradeoff-based models of the Internet”. *Proc. of SIWN/IEEE International Conference on Complex Open Distributed Systems*, volume 7. Citeseer, 2008.
- [139] Sperotto, A., G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. “An Overview of IP Flow-Based Intrusion Detection”. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010. ISSN 1553-877X.
- [140] Talbi, E.G. *Metaheuristics: from design to implementation*. Wiley, 2009. ISBN 0470278587.
- [141] Tan, KC, T.H. Lee, D. Khoo, and EF Khor. “A multiobjective evolutionary algorithm toolbox for computer-aided multiobjective optimization”. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(4):537–556, 2002. ISSN 1083-4419.
- [142] Vapnik, V., S.E. Golowich, and A. Smola. “Support vector method for function approximation, regression estimation, and signal processing”. *Advances in Neural Information Processing Systems 9*. Citeseer, 1996.
- [143] Vapnik, V.N. *The nature of statistical learning theory*. Springer Verlag, 2000. ISBN 0387987800.
- [144] Vapnik, V.N. “An overview of statistical learning theory”. *Neural Networks, IEEE Transactions on*, 10(5):988–999, 2002. ISSN 1045-9227.
- [145] Varga, A. et al. “The OMNeT++ discrete event simulation system”. *Proceedings of the European Simulation Multiconference (ESM2001)*, 319–324. 2001.
- [146] de Vivo, M., E. Carrasco, G. Isern, and G.O. de Vivo. “A review of port scanning techniques”. *ACM SIGCOMM Computer Communication Review*, 29(2):41–48, 1999. ISSN 0146-4833.
- [147] Wagner, A., T. D
”ubendorfer, B. Plattner, and R. Hiestand. “Experiences with worm propagation simulations”. *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 34–41. ACM, 2003. ISBN 1581137850.
- [148] Wagner, A. and B. Plattner. “Entropy based worm and anomaly detection in fast IP networks”. 172 – 177. jun. 2005. ISSN 1524-4547.
- [149] Waxman, BM. “Routing of Multipoint Connections”. *Selected Areas in Comm*, 6(9):1617–1622, 1988.
- [150] Weaver, Nicholas, Vern Paxson, Stuart Staniford, and Robert Cunningham. “A taxonomy of computer worms”. *Proceedings of the 2003 ACM workshop on*

Rapid malware, WORM '03, 11–18. ACM, New York, NY, USA, 2003. ISBN 1-58113-785-0. URL <http://doi.acm.org/10.1145/948187.948190>.

- [151] Willinger, W., D. Alderson, and J. Doyle. “Mathematics and the Internet: a source of enormous confusion and great potential”. *Notices of the American Mathematical Society*, 56(5):586–599, May 2009.
- [152] Willinger, W. and V. Paxson. “Where mathematics meets the Internet”. *Notices of the American Mathematical Society*, 45(8):961–971, 1998. ISSN 0002-9920.
- [153] Winick, J. and S. Jamin. *Inet-3.0: Internet Topology Generator*. Technical Report UM-CSE-TR-456-02, Department of EECS, University of Michigan, 2002.
- [154] Yook, S.H., H. Jeong, and A.L. Barabási. “Modeling the Internet’s large-scale topology”. *Proceedings of the National Academy of Sciences of the United States of America*, 99(21):13382, 2002.
- [155] Zacharia, G. and P. Maes. “Trust management through reputation mechanisms”. *Applied Artificial Intelligence*, 14(9):881–907, 2000. ISSN 0883-9514.
- [156] Zalewski, M. “Security engineering: broken promises”. Zero Day, May 2010. URL http://www.zdnet.com/blog/security/security-engineering-broken-promises/6503?tag=mantle_skin;content.
- [157] Zegura, E.W., K.L. Calvert, and M.J. Donahoo. “A quantitative comparison of graph-based models for Internet topology”. *IEEE/ACM Transactions on Networking (TON)*, 5(6):770–783, 1997. ISSN 1063-6692.
- [158] Zeng, X., R. Bagrodia, and M. Gerla. “GloMoSim: a library for parallel simulation of large-scale wireless networks”. *ACM SIGSIM Simulation Digest*, 28(1):154–161, 1998. ISSN 0163-6103.
- [159] Zhang, R., D. Qian, C. Ba, W. Wu, and X. Guo. “Multi-agent based intrusion detection architecture”. *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference on*, 494–501. IEEE, 2002. ISBN 0769513816.
- [160] Zhao, Qi, Jun Xu, and A. Kumar. “Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation”. *Selected Areas in Communications, IEEE Journal on*, 24(10):1840–1852, 2006. ISSN 0733-8716.
- [161] Zhu, Z., G. Lu, Y. Chen, Z.J. Fu, P. Roberts, and K. Han. “Botnet research survey”. *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, 967–972. IEEE, 2008. ISSN 0730-3157.

- [162] Zhu, Z., Y.S. Ong, and J.L. Kuo. “Feature Selection Using Single/Multi-Objective Memetic Frameworks”. *Multi-Objective Memetic Algorithms*, 111–131, 2009.
- [163] Zou, C.C., W. Gong, and D. Towsley. “Code red worm propagation modeling and analysis”. *Proceedings of the 9th ACM conference on Computer and communications security*, 138–147. ACM, 2002. ISBN 1581136129.
- [164] Zou, C.C., W. Gong, and D. Towsley. “Worm propagation modeling and analysis under dynamic quarantine defense”. *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 60. ACM, 2003. ISBN 1581137850.
- [165] Zseby, T., T. Hirsch, and B. Claise. “Packet sampling for flow accounting: Challenges and limitations”. *Passive and Active Network Measurement*, 61–71, 2008.

Vita

Capt David L. Hancock was commissioned in the Air Force in 2004 through the Reserve Officers Training Corps (ROTC) program, Detachment 860, at Utah State University, where he earned his Bachelor of Science degree in Computer Engineering. In 2006, Capt Hancock served as the chief of the Process Improvement branch in the U.S. Air Forces Europe (USAFE) Network Operations and Security Center (NOSC). From 2007-2009, Capt Hancock served with the 29th Intelligence Squadron and provided network security coordination and planning services to the National Security Agency (NSA) Threat Operations Center (NTOC), earning the Global Information Assurance Certification (GIAC) of Certified Incident Handler (GCIH) in the process. Capt Hancock has been awarded the Air Force Commendation Medal and the Joint Forces Commendation Medal. Upon graduation, Capt Hancock will be assigned to the 315th Network Warfare Squadron at Fort George G. Meade, Maryland.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 24-03-2011		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Jun 2009 — Mar 2011	
4. TITLE AND SUBTITLE A Multi Agent System for Flow-Based Intrusion Detection Using Reputation and Evolutionary Computation				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Hancock, David L., Capt, USAF				5d. PROJECT NUMBER 11-344	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/11-02	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Software Protection Initiative AFRL/RYT (Lt Col Kenneth S. Edge) Wright-Patterson AFB, OH 45433 937-320-9068 x181, kenneth.edge@wpafb.af.mil				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RYT	
11. SPONSOR/MONITOR'S REPORT NUMBER(S)					
12. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The rising sophistication of cyber threats as well as the improvement of physical computer network properties present increasing challenges to contemporary Intrusion Detection (ID) techniques. To respond to these challenges, a multi agent system (MAS) coupled with flow-based ID techniques may effectively complement traditional ID systems. This paper develops: 1) a scalable software architecture for a new, self-organized, multi agent, flow-based ID system; and 2) a network simulation environment suitable for evaluating implementations of this MAS architecture and for other research purposes. Self-organization is achieved via 1) a "reputation" system that influences agent mobility in the search for effective vantage points in the network; and 2) multi objective evolutionary algorithms that seek effective operational parameter values. This paper illustrates, through quantitative and qualitative evaluation, 1) the conditions for which the reputation system provides a significant benefit; and 2) essential functionality of a complex network simulation environment supporting a broad range of malicious activity scenarios. These results establish an optimistic outlook for further research in flow-based multi agent systems for ID in computer networks.					
15. SUBJECT TERMS Intrusion Detection, Multi Agent Systems, Network Security					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)
U	U	U	U	178	Dr. Gary B. Lamont (937) 255-3636, x4718; gary.lamont@afit.edu