AFRL-RI-RS-TR-2017-064

BIO-INSPIRED DISTRIBUTED DECISION ALGORITHMS FOR ANOMALY DETECTION

RUTGERS UNIVERSITY

MARCH 2017

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2017-064 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
ROBERT KAMINSKI
Work Unit Manager

/ S /
WARREN H. DEBANY, JR.
Technical Advisor, Information
Exploitation and Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MARCH 2017	FINAL TECHNICAL REPORT	SEP 2012 – SEP 2016

4. TITLE AND SUBTITLE

BIO-INSPIRED DISTRIBUTED DECISION ALGORITHMS FOR ANOMALY DETECTION

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-12-2-0232

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Nina H. Fefferman

5d. PROJECT NUMBER
DHS2

5e. TASK NUMBER
RU

5f. WORK UNIT NUMBER
TG

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rutgers University
New Brunswick, NJ 08901

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIG
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2017-064

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research effort brought together computer scientists and biologists to investigate the potential for self-organizing anomaly detection protocols inspired by those observed naturally in colonies of social insects to provide appropriate, dynamic, detection thresholds for anomalous event patterns on computer system networks to improve early detection and rejection methods to counter malicious threats.

15. SUBJECT TERMS
DIAMoND, Local Anomaly Detector, Total Impact Estimation, Threat Level Estimator

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**ROBERT L. KAMINSKI**
U	U	U	UU	53	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

LIST OF FIGURES

Section **Page**

LIST OF TABLES

1.0 SUMMARY

This goal of this research effort was to bring together computer scientists and biologists to investigate the potential for self-organizing anomaly detection protocols inspired by those observed naturally in colonies of social insects to provide appropriate, dynamic, detection thresholds for anomalous event patterns on computer system networks to improve early detection and rejection methods to counter malicious threats. The research relied on basic algorithmic development, based on bio-inspired theory, from researchers in both biology and computer science at Rutgers University, and also on hardware implementation, testing, and real-world tech transition development by partners at Honeywell International Incorporated. Simulation testbeds were constructed in both software and hardware. Social insect colonies have survived over evolutionary time in part due to the success of their collaborative methods, using local information and distributed decision making algorithms, to detect both positive and negative anomalies in their environment, thereby identifying critical resources and avoiding dangerous threats. These methods have the unusual and useful ability to detect anomalies with very little memory and using only very local information. Relying on insights from these systems, we analyzed designed initial algorithms based on both bees and ants. Early results rapidly converged on a single bio-inspiring system (honeybees were quickly seen to provide better results than ants) and ongoing efforts then tested/tailored the algorithmic design on a variety of attacks, normal patterns in background network traffic, and underlying network topologies. Results from these tests led to algorithmic revisions, and the process was iterated. The resulting algorithm has been presented in a series of 4 accepted peer-reviewed publications in highly respected conferences/journals, one pending provisional patent application, and an already-funded grant for further research into the utility of the developed algorithm in other contexts by the National Science Foundation.

2.0 INTRODUCTION

Early and accurate detection of anomalies in patterns of network traffic allow the best chance for mitigation of threats to sufficient/stable network function. Whether these threats are purposeful attacks or accidental, cascading failures, identification of divergence from (sometimes highly variable) “normal traffic” is critical. The most traditional methods in mathematics, engineering, operations research, and computer science for designing surveillance networks have focused on hierarchical structure, where local observations are passed up a command chain, eventually reaching a centralized gathering of intelligence for broad-view analysis. These methods allow for in-depth synthesis and analysis, considering all the available information at once, but also involve a variety of problematic features. They create a communications burden on the system, cause a delay in detection while information is being gathered from the different sources and synthesized, and require at least some level of trust in both participants and communications security to enable the sharing of the potentially sensitive information being analyzed along all the links in the chain from observer to central analyst. On the other extreme, truly parallel, distributed methods are fast and private, but can observe and analyze only local information, thereby failing to see the ‘big picture’ as they focus on only one thread in a tapestry. Further, regardless of central vs. parallel analysis, existing methods typically rely on a statistical or rule-based model of the expected behavior of network traffic, and anomalous activity is identified as any behavior that is not

adequately explained by the model [1]. However, these models will fail to capture threats which purposefully and maliciously masquerade as 'normal network behavior.'

Social insects in the natural world routinely need to make classification decisions, and take actions as a result of those classifications, rapidly and accurately to ensure their survival. As with the desired features of network surveillance systems, both false positives and false negatives equally compromise success [2]. Critically, these natural systems also include features that, while currently mostly ignored in network surveillance design, are likely to be highly desirable in future generations of network defense: they easily integrate information from multiple sources at once, and are able to make accurate distributed decisions under shifting baseline conditions. Species in nature that focus only on the most obvious threat, or the richest food source, survive poorly since the natural world includes a simultaneous diversity of threats and more often than not, the richest food source is still not sufficient to sustain the population without supplementation from additional sources. For these reasons, anomaly detection algorithms found in social insect systems are already well adapted to detecting multiple anomalies at once, providing the ability to focus attention specifically on one particular facet of observed data without losing the ability to continue to scan on a coarser-scale for additional threats.

To try and leverage the insights in anomaly detection systems available from the study of social insects in the natural world, and apply those in a way that would provide rapid, scalable, and practical methodologies for human cyber-infrastructure systems, we developed a new algorithm: Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection (DIAMoND). DIAMoND employs distributed surveillance, but instead of fully independent, parallel analysis, each observer/analyst incorporates into their own conclusions the nonparametric descriptions of conclusions shared by their network neighbors. This algorithm achieves the rapid, real-time detection time of fully parallel surveillance, solves the problem of integrating information coming from different scales or gathered by different methods, and eliminates the need to share sensitive details/data of local observations or analysis across potentially insecure channels. Our efforts involved both software and hardware implementations testing DIAMoND's performance at detecting a variety of threats (focused primarily on DDoS and Stealth Scan attacks), over a variety of baseline (e.g. non-anomalous) network traffic patterns, and across a diversity of simplified and realistic network topologies. The tests showed DIAMoND achieved improvement in early and accurate detection relative to both purely parallel and purely centralized approaches (i.e. methods currently employed in modern network anomaly detection). To investigate the practicality of integrating DIAMoND into existing surveillance networks, we performed software simulation-based tests to study the impact of only partial network deployment, and found that 30% deployment among surveillance nodes would be sufficient to gain over 80% benefit in anomaly detection improvement. Our research also extended DIAMoND to consider cases in which the DIAMoND network itself contained participating malicious actors, and test its application as a DNS-based mitigation system.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

Our research involved an iterated process of theoretical design (based on natural systems and algorithm performance under testing) (Rutgers), and both software (Rutgers) and hardware (Honeywell) simulation testing. We will therefore describe each of these phases to our process separately, but they should be understood as repeated steps.

3.1 Theoretical Algorithmic Design Based on Social Insect Systems

The basis for the algorithmic design comes from an understanding of social insect systems and the ability to describe such systems using only mathematical logic, rather than biological or English language descriptions. We therefore characterized the features of independent identification of potential anomalies, communication of independent identification (in)decisions among agents (i.e. insects), and the dynamics of updating individual decision making algorithms based on the shared insights from communication.

Abstraction of the logic of the system led us to design a simple, method for leveraging non-parametric information about independent, parallel parametric analyses. See Figure 1.

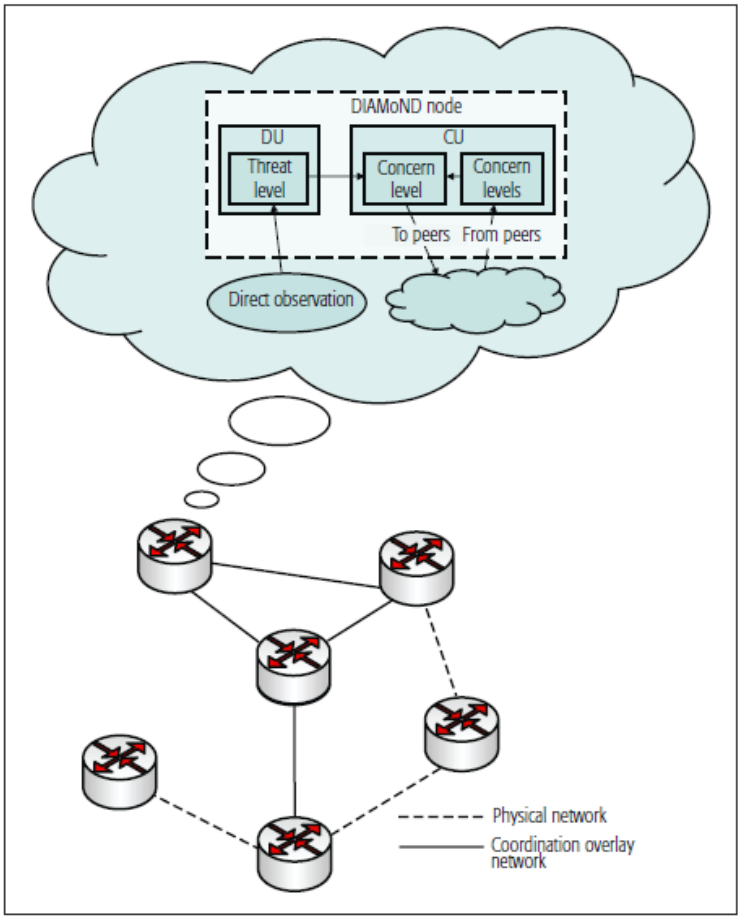

Figure 1. DIAMoND Abstract Design Logic

Mathematically, our DIAMoND population, consists of nodes $b_1 \ldots b_x$ and a computer network (i.e. an edge set, $E = \{e_1 \ldots e_y\}$) on the nodes where each edge is a pair b_i, b_j such that $i \neq j$.

Each node has access to some subset of information about the packets it handles, we call this the *Observation set* of the node, $o(b_i)$. This may be, but is not assumed to be the same set across all nodes. For each node, there is also a time *window*, $W_i^{t-w,t}(o(b_i))$, of length *w* over which the

observation set may be analyzed (note: this may be interpreted as an iterative function, such as a check sum, but may also be interpreted as loosely as storage of the full packet stream information over the node for a duration of time; if there is no limit on memory, $w=t$).

Each node also has an internal set of *anomaly detection* algorithms, $a_i\left(W_i^{t-w,t}\left(o\left(b_i\right)\right), st_{i,t}\left(b_i\right)\right)$ that are a function of the information in the observation set of that node and also of the set of detection *sensitivity thresholds* for that node at time *t*, $st_{i,t}\left(b_i\right)$, each associated with a detection algorithm available to that node (i.e. each element of st_i is associated with an element of a_i). These sensitivity thresholds may be either static or dynamically updated over time, and there is no *a priori* assumption of uniformity in sensitivity thresholds across nodes.

At each time *t*, each node computes a function of the observed *threat level*, $T_{i,t}\left(a_i\right)$, which is the data-driven assessed level of likelihood that an anomaly is occurring.

Each node *i* also has an associated set of nodes that influence the sensitivity threshold of *i* called the *neighborhood*, $n(b_i) \supseteq N$, such that $b_j \in n(b_i)$ st $b_i, b_j \in E$. Further, each element of the neighborhood is associated with an edge weight as a relative *measure* of their influence on the sensitivity threshold of *i* at time *t*, $m_{i,t}(b_j)$ for all $b_j \in n(b_i)$. Note, these measures of influence can either be static or can be dynamically updated over time.

Each node *i* has a level of *concern* at time *t*, $c_{i,t}\left(T_{i,t-1}, L_{t-1}\left(n\left(b_i\right)\right)\right)$, which is a function of both the previously assessed threat level and of a function $L_t\left(n\left(b_i\right)\right)$ that computes the total impact of the concerns of all nodes within the neighborhood of *i* at time t.

Our Bee-Inspired DIAMoND Algorithm:

Initially, we define $c_{i,1} = 0$ and $L_1\left(n\left(b_i\right)\right) = 0$ for all *i*. We set

$$L_t\left(n\left(b_i\right)\right) = \begin{cases} 0 & if \quad \dfrac{\sum_{b_j \in n(b_i)} c_{j,t-1}}{\left|n\left(b_i\right)\right|} < 0.034 \\ 1 & if \quad 0.034 \le \dfrac{\sum_{b_j \in n(b_i)} c_{j,t-1}}{\left|n\left(b_i\right)\right|} < 1.34, \\ 2 & if \quad 1.34 \le \dfrac{\sum_{b_j \in n(b_i)} c_{j,t-1}}{\left|n\left(b_i\right)\right|} \end{cases} \tag{1}$$

(where the current values listed were informed by the uniform average degree of our initial network topology).

We define a_i for all *i* as the accurate sampling scheme based on Korczynski, Janowski & Duda [5], with adaptive sensitivity thresholds, st_i, such that

$$st_{i,t} := \begin{cases} st_{i,t-1} + c & if \quad L_t\left(n\left(b_i\right)\right) > L_{t-1}\left(n\left(b_i\right)\right) \\ st_{i,t-1} & if \quad L_t\left(n\left(b_i\right)\right) = L_{t-1}\left(n\left(b_i\right)\right) \\ st_{i,t-1} - c & if \quad L_t\left(n\left(b_i\right)\right) < L_{t-1}\left(n\left(b_i\right)\right) \end{cases}, \tag{2}$$

where *c* is some constant.

We then assign $T_{i,t}(a_i) \in \{0,1,2\}$ for each node in each time based on the observed traffic on that node using algorithm a_i with thresholds $st_{i,t}$.

We then defined four different test functions $c_{i,t}$ for all nodes in order to explore the impact of different levels of importance being assigned to the concern of neighbors (low, med, med+, and high), such that the function $c_{i,t}(T_{i,t-1}, L_{t-1}(n(b_i)))$ is described according to Table 1 immediately below.

Table 1. Test Functions at Concern Levels

$T_{i,t}(a_i)$	$L_t(n(b_i))$	$c_{i,t}$			
		low	med	med+	high
0	0	0	0	0	0
0	1	0	0	1	1
0	2	1	1	1	2
1	0	1	1	1	1
1	1	1	1	1	1
1	2	1	2	2	2
2	0	2	2	2	2
2	1	2	2	2	2
2	2	2	2	2	2

In the implementation of this naïve algorithm, nodes send information on their levels of concern to their neighbors for inclusion in independent determination of threshold violations for parallel anomaly detection.

Each of the described communication frequencies, thresholds for incorporation of information from neighbors, and determination of levels of concern based on independent threshold violation were then refined under iterated testing to produce our current algorithm (as described in [3]; See Appendix B for paper presenting full details).

3.2 Software Simulation Testing

3.2.1 Simulation of Network Traffic

3.2.1.1 Mininet and OpenFlow to Simulate Realistic Network Traffic. We developed a prototype communication controller as an OpenFlow component in the POX environment and evaluated in Mininet 2.0 simulator. We developed and continue to support POX controllers, prototype implementation of the communication protocol (described in 3.1 above), and testing scenarios.

We designed OpenFlow switches to act as a "standard" type of L2 learning switch analogically to forwarding.l2_learning (as described in [4]). However, it ignores the distributed-detection communication packets (Ethernet type 0x0105) that are exchanged between switches participating in the DIAMoND network. This component detects all kind of anomalies related to the TCP protocol locally (as described in [5]). In first place, it creates neighborhoods (smaller subnetworks within which the information is exchanged) based on hop limits. Second, it combines levels of concern of its neighbors with a report from its internal anomaly detection system.

Full details and example simulation commands can be found in Appendix A below.

3.2.1.2 Simplified Network Traffic. We defined 'regular' traffic gi(*t*) in a node i over time follows a given distribution. The assumption is that each node had been monitoring the traffic that it handles under normal conditions. In our simulations, this distribution has the same functional form for all nodes in a given realization. However, since different amounts of traffic go through each node, every node uses different values for the distribution parameters (and these do not change over time). The simulation proceeds at discrete time steps. At each time step we choose a random number from the given distribution for a node, according to this node's stored parameters. This random number includes all the regular (non-attack) traffic that goes through this node at the given time step, and we do not explicitly follow the path of individual normal traffic packets. Therefore, normal traffic is uncorrelated even between neighboring nodes. We use three different forms of normal traffic distributions, which correspond to different scenarios: a) Gaussian, b) uniform, and c) exponential.

Gaussian: In this case, the regular traffic in a node is normally distributed. The expected value of traffic (in arbitrary units) in each node is constant with time, and uniformly distributed in the range μ:[750:1250]. The standard deviation for each node is also constant with time, and is uniformly distributed in σ:[25:100]. So, at time step *t,* the traffic gi(t) on each node i is selected from a gaussian distribution (μi,σi).

Uniform: The expected traffic value for a node is chosen uniformly in the range μ:[750:1250]. The width of the distribution is a random number in the range σ:[25:100]. This means that at time step *t* the traffic gi(t) on a node is uniformly selected in the range [μi-σi:μi+σi].

Exponential: In this case a random number is drawn from an exponential distribution λexp(-λr) with parameter λ=1/β, where β is chosen uniformly in the range β:[750:1250].

3.2.2 Simulation of Diversity of Network Topologies. Due to the complexity of testing the emergent outcomes of the algorithm, we chose to split the testing into two scenarios: (1) realistic underlying patterns in normal network traffic (as described in section 3.2.1 above) on simplified network topologies, and (2) simplified underlying patterns in normal network traffic on realistic network topologies.

3.2.2.1 Simplified Physical Topologies. In order to explore the impact of network configuration on the success of the distributed detection of DIAMoND, we tested the performance of the algorithm on a variety of network topologies (each with a total network size of 20 nodes due to computational constraints). These included "Full Mesh", "Mesh", "Bipartite", "Linear", and "Extended Star", defined as follows:

"Full Mesh" is the complete undirected network on 20 nodes, involving n*(n-1)!/n=190 edges.

"Mesh" is a randomly selected subgraph of Full Mesh in which the probability of each edge is 0.3, therefore involving 0.3*n*(n-1)!/n=57 edges for each realization.

"Bipartite" is the division of the network into two sets of size 10, with the probability of 0.6 for an edge between nodes from different sets and a probability of 0 for an edge between nodes in the same set, therefore also involving 0.6*n*(n-1)!/2n=57 edges for each realization.

"Linear" is a single path of edges among the nodes, i.e. a connected graph in which 2 nodes have exactly 1 incident edge and all others have exactly 2 incident edges, therefore involving (n-1)=19 edges.

"Extended Star" is a tree, created by initiating the graph with 1 node and then attaching each subsequently created node to one of the already existing nodes with uniform probability until the network size reaches 20, therefore involving (n-1)=19 edges.

3.2.2.2 Realistic Physical Topologies. In our simulations, we use six different model network topologies: a) a two-dimensional square lattice, where nodes occupy the vertices of a lattice, b) an Erdos-Renyi network [6], where nodes are connected randomly to $<k>=3$ other nodes, c) a scale-free network [7] created with the configuration model [8], where the degree distribution follows a power law with degree exponent $\gamma=3.5$, d) a scale-free network with degree exponent $\gamma=2.5$, where by definition the hubs are much stronger, i.e. they connect to a larger number of nodes, e) the CAIDA Autonomous System graph for May 2004 [9], and f) the CAIDA Autonomous System graph for May 2007 [9].

3.3 Hardware Simulation Testing

3.3.1 Assumptions for Initial Hardware Testbed (later abandoned). The design here described was abandoned by Honeywell before construction due to loss of lead researcher and insufficient funds to reconstruct efforts after the fact. Instead, Honeywell later focused on design and testing of a DNS-side attack mitigation strategy (see section 3.3.3 below).

Given the generic nature of the DIAMoND algorithm and the problem DIAMoND is expected to solve, we make the following assumptions when designing our testbed:

1. Emulation on Packet exchange level on network layer is sufficient.
    - No need for PHY and MAC emulation
    - No need for buffer behavior manipulation at least for Phase I
2. Time synchronization precision on ms level is sufficient
    - Common Internet packet delivery delay on ~10ms level so <1ms precision is good enough for us to properly maintain order of events.
3. Linux user space applications are sufficient for DIAMoND algorithms emulations
    - Given that ms level delay/time sync errors are acceptable, use space applications are expected to be sufficient for real-time processing.
    - Linux packet queue manipulation tools are available if Kernel configuration for packet processing is needed (e.g. netem)
    - Command line tools such as iptable can be used to dynamically reconfigure how kernel handles network traffic

3.3.2 Initial Hardware Testbed Design (later abandoned). The design here described was abandoned by Honeywell before construction due to loss of lead researcher and insufficient funds

to reconstruct efforts after the fact. Instead, Honeywell later focused on design and testing of a DNS-side attack mitigation strategy (see section 3.3.3 below).

3.3.2.1 Hardware Architecture. Figure 2 illustrates the hardware setup architecture. 21 machines running Linux 12.0.4 LTS were planned. Every machine was to have 2 Ethernet ports (Eth0, Eth1). All Eth0 ports were to be connected to a dedicated routers and were to have been Intenet accessible with IP address of 131.201.16.(175+nodeID). All Eth1 ports were to be connected to another dedicated router to form a private LAN with IP address of 192.168.1.(nodeID). The Internet acceissble ports were to be utilized for adminatration and testing management purposes. This private LAN was to have been dedicated to simulation of network traffic and execution of the DIAMoND alagorithm.

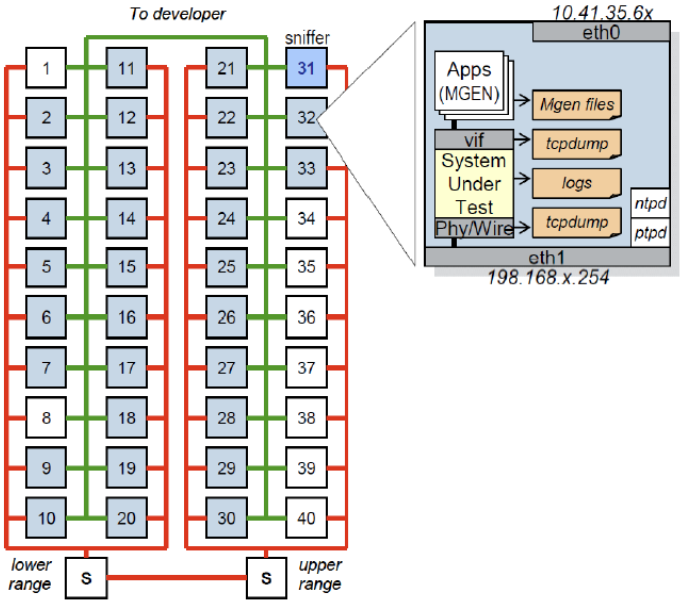

Figure 2. DIAMoND Hardware Architecture

Eth1 on Node 21 was to be connected to port 48 on Cisco 3650 router, which was to have been set up to be a SPAN port for network traffic monitoring. Network Time Protocol (NTP) would have been set up on all machines with Node 21 time sync to Honeywell time synch master. All other machines were to time sync to Node 21 as a local master.

3.3.2.2 Network Setup. Figure 3 shows the intended network setup for node i. Each node was to have represented a large network address space. Figure 3 illustrates a sample /16 network setup, simulating a network router managing 65535 hosts. The DIAMoND Virtual Network Service (VNS) was to have been capable of "spoofing" any arbitrary source IP address, intercept packets destined to any Node i's configured subnet. VNS was also to have been responsible for setting up raw IP packet, UDP packet and TCP sessions including handshaking and teardown. DIAMoND

Traffic generators, DIAMoND agents and local logging services were all to go through BNI (DIAMoND Network Interface) to access information on the simulated network.

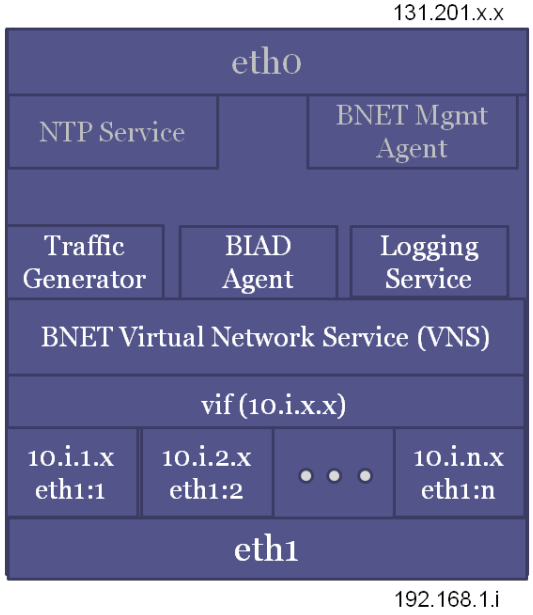

Figure 3. Sample Network Setup for Node i

3.3.2.3 Software Architecture

Software modules were to be grouped into three categories: System Services, DIAMoND agents, Traffic Generations Services and DIAMoND management services. See Figures 4 and 5.

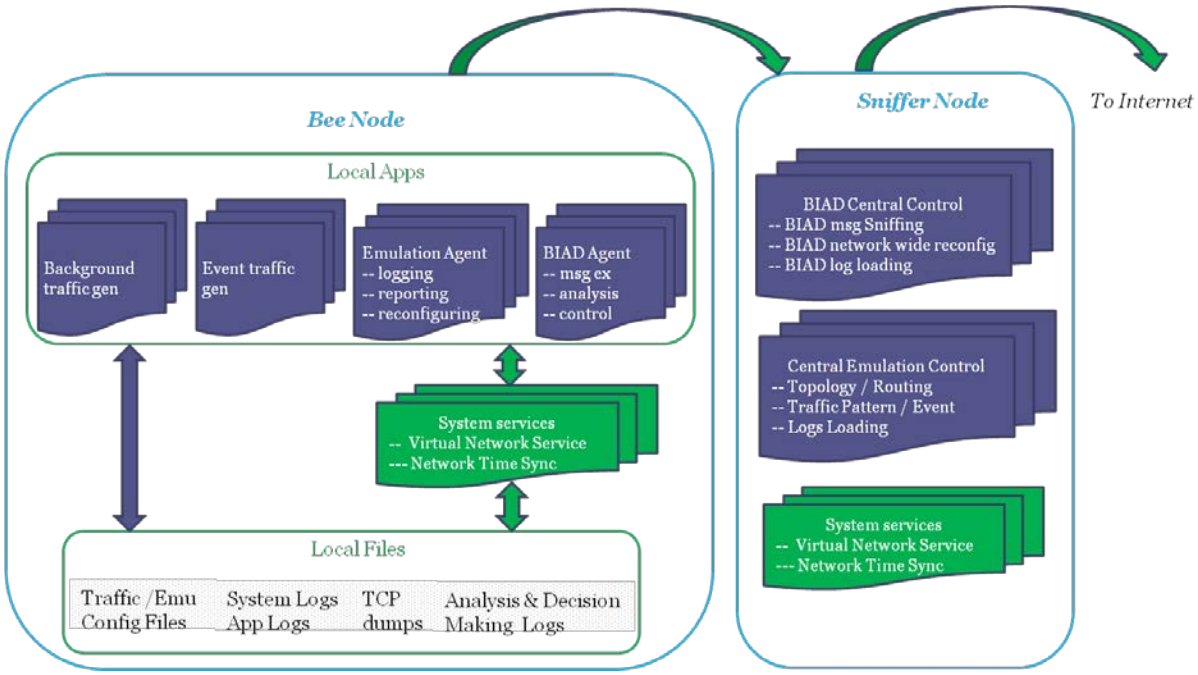

Figure 4. DIAMoND Software Architecture

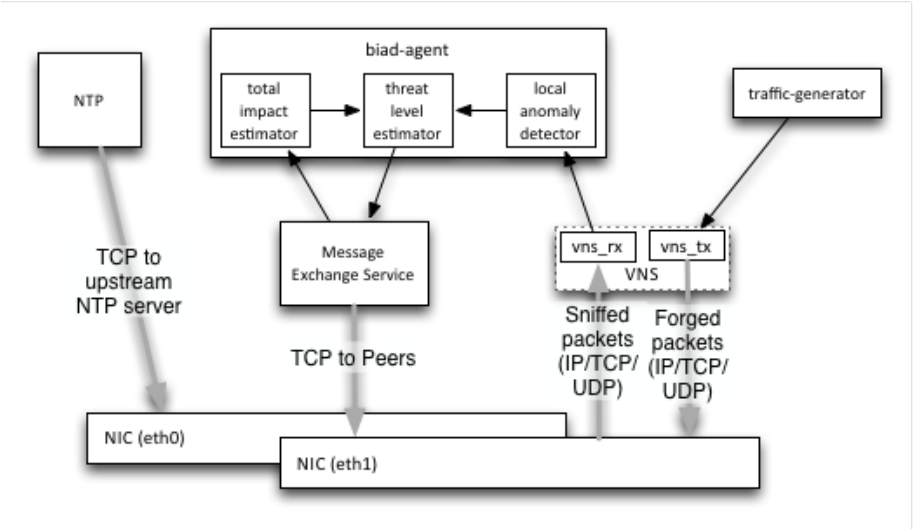

Figure 5. DIAMoND Node Software Architecture

3.3.2.4 System Services. System services were to have included the following modules:

1. Virtual Network Services (VNS)
2. Network Time Synchronization Services (NTS)

We were to have used Linux NTP (version number) service for this purpose. Network Time Protocol (NTP) would have been set up on all machines with Node 21 time sync-ed to Honeywell

time synch master. All other machines were to have been time sync-ed to Node 21 which serves as a local master.

VNS was to have utilized *Scapy* libraries for rapid raw packet construction and *libpcap* for direct packet interception on ethernet cards. VNS was supposed to completely bypass the traditional Linux network stacks. When Kernel cooperations are needed, iptable tools were to have been utilized as illustrated in Network Setup section.

The functional spec of VNS is summarized below:

1. VNS should provide a set of APIs to
    a. build and transmit an raw IP packet with any specified (srcIP, destIP) and payload.
    b. intercept and respond to incoming IP packets within specified timeline.
        i. E.g. packets to a virtual host that does not physically exist but virtually configured and managed by the receiving node
    c. filter incoming IP packets and pass summarized states to callers

2. VNS should provide a set of APIs to
    a. build a UDP packet with proper protocol flags and specified (srcIP, srcPort, destIP, destPort) and payload.
    b. transmit a UDP packet using the IP APIs decsribed in 1.
    c. intercept and respond to incoming UDP packets within specified timeline.
        i. E.g. packets to a virtual host that does not physically exist but virtually configured and managed by the receiving node
    d. filter incoming UDP packets and pass summarized states to callers

3. VNS should provide a set of APIs to
    a. Initiate and tear down TCP sessions with a specified (srcIP, srcPort, destIP, destPort) (i.e., spoofing the src address.)
    b. build a TCP packet with proper protocol flags and specified payload.
    c. transmit a TCP packet using the IP APIs decsribed in 1.
    d. intercept and respond to incoming TCP packets within specified timeline.
        i. E.g. packets to a virtual host that does not physically exist but virtually configured and managed by the receiving node
    e. filter incoming TCP packets and pass summarized states to callers

3.3.2.5 Diamond Agents. DIAMoND agents are responsible for simulating “DIAMoND node” behaviors with flexible LAD, TIE and TLS algorithms. DIAMoND agents include the following modules:

1. DIAMoND Message Exchange Service (MES)
2. Local Anomaly Detector (LAD)
3. Total Impact Estimation (TIE)
4. Thread Level Estimator (TLS)

Functional specs for MES were to be:

1. Takes a pre-specified dynamic network topology configuration file and establish direct connections (TCP) between every pair of nodes which are connected per the topology
2. Every node is able to transmit messages asynchronously to all of its direct neighbors
3. Invoke LAD via pre-specified APIs with new threshold to compute local anomaly score
4. Invoke TIE via pre-specified APIs to compute total impact from its neighbors based on received messages on neighbors' levels of concern.
5. Invoke TLS via pre-specified APIs to compute the local node's level of concern and update its neighbors
    a. Level of concern is updated only when there is change from last update

3.3.2.6 Traffic Generation Services. System services include the following modules:

1. Background Traffic Generation Services (BTG)
2. Event Traffic Generation Services (ETG)
3. Replay of Traffic Traces (RTT)

BTG creates "norm" traffic background with pre-specified distribution, BTG takes in a configuration file and generate packets pertaining to specified distributions. BTG utilizes VNS to interact with physical interfaces. As a starting point, BTG supports only uniform packet distribution with variable rates and durations. More complex patterns will be supported later or approximated with piece-wise uniform pattern.

ETG creates "abnormal" traffic with pre-specified timing, type configurations. As a starting point, ETG will support random port scan (both horizontally and vertically).

RTT reads summarized network trace files and regenerate traffic pertaining to the traces. RTT is cutom-built to specific network traces. We plan to obtain and support re-play of Witty Worm traces from CAIDA managed by UCSD.

3.3.2.7 DIAMoND Management Services (BMS). BMS handles all administrative and automation tasks for setting up DIAMoND, administrating testing scenarios, logging and analyzing data in real-time.

Functional specs for BMS are:

1. Load a new testing scenarios (topology, traffic trace/configuration, DIAMoND algorithms) and distribute to all nodes
2. Start and shut down a testing scenario simultaneously on all nodes
3. Retrieving logs from all nodes (e.g. level of concern, local anomaly score changes over time)
4. Visualize level of concerns of all nodes

3.3.2.8 Traffic Generation. In order to generate traffic (background, simulated attacks, etc.) use the traffic generator located in ^/Software/traffic-generator/traffic_generator.py. The generator

relies on configuration files in the same folder, and is started with a configuration file as an argument.

3.3.2.9 DIAMoND Validation Test Scenarios

3.3.2.9.1 Port Scan Test Scenario. Design of engineering scenarios was to have followed these guidelines:

- Driven by real world scenarios
- Allow systematic evaluation of DIAMoND performance
- Our focus is on distributed collaboration among routers. We will assume known local anomaly detection mechanism
- Share common and practical metrics that are applicable in real world scenarios
- Enable us to pick right real world applications

Horizontal TCP service Port Scan was to have been the first scenario to mirror software simulation choices by the Rutgers researchers. The port scan event in our 20-node network was to have been designed to start from a single malicious node with fixed IP and arbitrary source port to reach multiple destination IPs with the same service port numbers. In the setup, the destination port numbers were to have been 80 and 443 reflecting the most popular TCP services for HTTP and SSH. The whole test scenario set consists of 25 individual tests with the DIAMoND network topology and service levels as control variable. The port scan attack remains the same for all scenarios.

3.3.2.9.2 Test Network Topologies. In this test set, we were to have considered the following 5 different DIAMoND network topologies to test how network connectivity affects DIAMoND collaboration performance:

- Topology 1: Linear Circular

Figure 6. Linear Circular Topology

- Topology 2: Fully Connected Mesh for all 20 nodes.

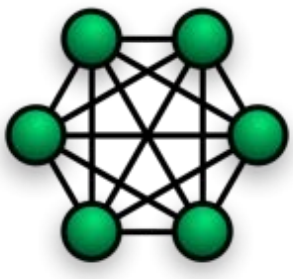

Figure 7. Full Connected Mesh with 6 Nodes

- Topology 3: Hierarchical 2 level structure with 5 hub nodes, illustrated below. All 5 hub nodes are fully connected while each cluster has a start topology with all lead nodes connected to a hub node.

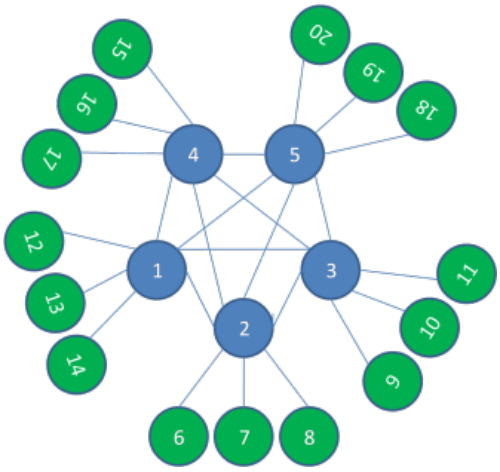

Figure 8. Hierarchical 2-Level Topology

- Topology 4: Hierarchical 2 level structure with 3 hub nodes. Similar to above except there are only 3 fully connected hub nodes and 3 clusters. Two of those clusters have 7 nodes and another one has 6 nodes.
- Topology 5: Hierarchical 2 level structure with 2 hub nodes. Similar to Topology 3 except there are only 2 fully connected hub nodes and 2 clusters. Each cluster has 10 nodes.

Topologies 1 and 2 represent the lower and upper bound cases in connectivity. Topologies 3 to 5 represent commonly used hierarchical network structures with varying number of gateways.

3.3.2.9.3 Service Levels. In this test set, we were to have considered the following 5 different service levels (0%, 10%, 20%, 30%, 50%, 80%) representing different types of operations. Service level is defined as percentage of the nodes inside the 20-node network that service port 443 or port 80. These nodes would have properly initiated a TCP session and exchanged data packets with any clients (both normal and attacker) probing port 443 and 80. All other nodes would not have responded to any requests to these two ports. These service level tests would have enabled us to understand how DIAMoND performs in different type of operating networks such as a corporate network with limited hosted services or a server farm such as Amazon EC2.

Varying service levels would also have allowed us to simulate attack stealthy-ness since higher service levels makes port scan harder to detect thus stealthier.

Varying the 5 different types of topologies and 5 different service levels would have given us 25 total test scenarios.

3.3.2.9.4 Traffic Setups. In all 25 test cases, the background (normal) traffic would have been set up as follows:

Each of the 20 nodes initiates a TCP session with all other 19 nodes on a pre-defined service port at 1Hz. Each TCP session would have been properly torn down after 2 normal data packets

exchanges. In total, there were to have been 19x19 successful TCP sessions going on inside the network every second.

In all 25 test cases, the event traffic was to have been set up as follows:

Attacker Node X, which would always have been a child node in all topologies, would start scanning port 443 on all nodes (including itself) from time t1 to t2 at a rate of 2Hz. T1 and t2 are configurable in DIAMoND configuration file. In our test set, they were to have been set at 5 seconds and 1500 seconds. Selection of these numbers would have ensured observations of two complete network wide state transitions: normal to abnormal; abnormal to normal. Selection of destination nodes was to have been fully random. In total Node X will try to initiate 2x20=40 TCP sessions within the test network. Note that the attacker probing rate has to be larger than the normal traffic rate because the local anomaly detector use SYN packet and ACK pack mismatching (instead of SYN and SYN-ACK mismatching) for anomaly detection. If normal traffic rate on a node is higher than probing rate, the local anomaly detector will never trigger any alarm regardless what the alarm threshold is. This is also one of the reasons that DIAMoND performance is so dependent on how the traffics are set up. In addition, the startup time for the attack plays a significant role in performance because the IP counter will keep going up with normal traffic alone given the way the local anomaly detector is designed / implemented. This "feature" forced us to add a cap on the IP counter to offset this artificial effect. For this reason, we also evaluated the dependence of DIAMoND performance on the IP counter cap.

3.3.2.10 Performance Evaluation Metrics. Given the local anomaly detector is based on TCP session negotiation protocols, it is natural to classify traffic on TCP session level. i.e. every TCP session will be labeled as either "normal" or "abnormal". Thresholds were to have been determined by experimental output from software simulation at Rutgers. By testing a diverse selection of specific thresholds, we expected to gain generalized insights of DIAMoND performance.

The DIAMoND performance was to have been evaluated using aggregated False Positive Rate (FPR) and False Negative Rate (FNR) on all nodes. In addition, we were also to have evaluated delay in first detection on network wide to characterize responsiveness of DIAMoND.

As a benchmark, we were to have used the same local anomaly detector without any coordination (i.e. fixed thresholds) for comparison.

3.3.3 Updated Hardware Simulation (replaced earlier efforts)

3.3.3.1 DNS Server-Based DIAMoND Mitigation Testing: Distributed Rate Sharing based Amplified DNS-DDoS Attack Mitigation (DRS-ADAM). We conducted our experiments in both software simulation environment and on a dedicated hardware testbed. Our software simulation code is built on a popular networking tool called Mininet (see detailed description in Appendix A) which creates a realistic virtual network environment by emulating real kernel, switch and application code, on a single machine. For machine, we used a DELL T320s server with a 2:4 GHz Intel Core 10 CPU and 16 GB of RAM, running VMware ESXi v5:5 virtualization software. An Ubuntu 14:04:1 LTS virtual machine was deployed on the ESXi server. To build a virtual network, Mininet uses network name spaces and virtual Ethernet pairs (veth) mechanisms. On the top of that we emulated bots, resolvers, and the victim servers which were later deployed on the same virtual machine, while each component having its own isolated stack of code, application, and network and was connected using a veth pair. To explore the functionality,

scalability and performance issues of DRS-ADAM beyond software simulation environment, we designed a hardware testbed. This is important in the regard that many solutions often overlooked the practical issues one might face in the real world. For building the hardware testbed we employed 20 machines with 2:8 GHz Intel core processor and 16 GB RAM, and deployed Ubuntu 14:04:1 LTS on each of them. A virtual environment is created using virtual IP addresses and virtual Ethernet interfaces which can be controlled by a central server. Each machine than becomes a hub for certain number of DNS resolvers and bots to reflect a real-world attack. To emulate wide area network topology, delays are configured using netem [10]- a tool to create sophisticated delays of desired distribution.

3.3.3.1.1 Emulated topology. The tested topology illustrated in Figure 9 was designed to emulate the attack traffic logical topology and the distributed nature of the defense mechanism. Bots and resolvers were organized in clusters; each cluster emulated an autonomous system. The size and the number of clusters for resolvers and bots varied depending on the setting for each test. Each cluster of bots and resolvers was attached to an edge router. These edge routers were connected through transit routers, and the added delay or latency between them was set to emulate an Internet link-aggregation/core. The resolvers edge router is connected to the victim's edge router also by the means of transit routers. There are three types of links; link between resolvers/bots and their edge router, link between transit routers and victim's link to its edge router.

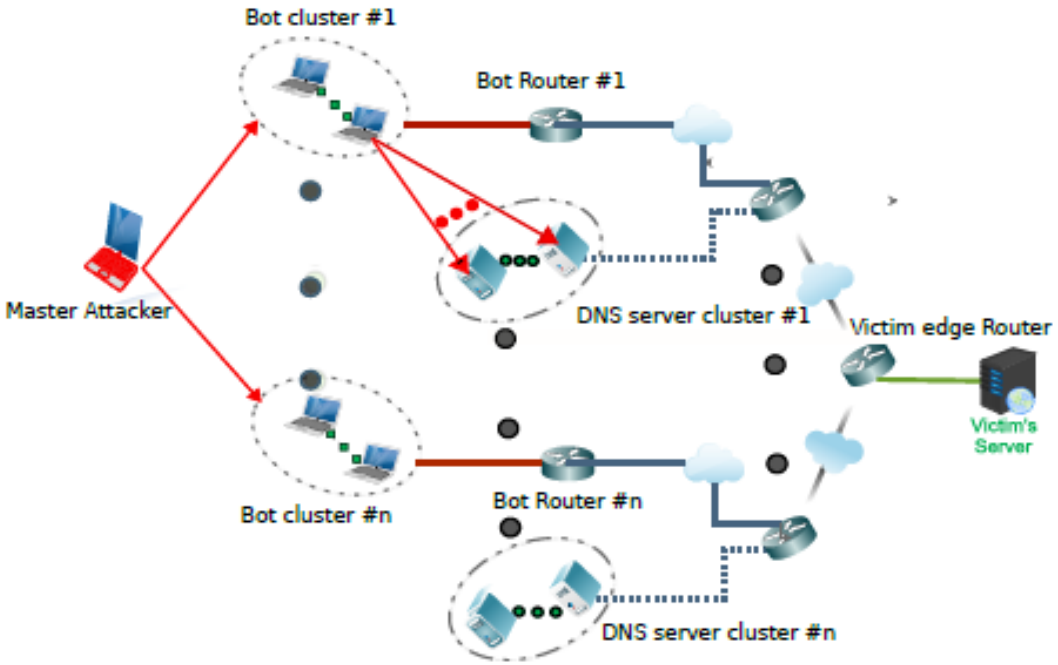

Figure 9. DDoS Attack

3.3.3.1.2 Attack traffic. Each bot in a cluster was programmed to send spoofed TXT queries to a designed cluster of resolvers.

The following is a subset of DRS-ADAM test instructions:

(i) set up and start the virtual network with the emulated topology; (ii) specify the hosts that will behave as bots or name servers, taking into consideration clustering; (iii) start the implemented

resolver detector module (see Section 4.1) on every name server-assigned machine; (iv) monitor name servers' exchange messages; (v) start the implemented victim monitor module at the assigned host; (vi) monitor victim exchange messages and link rates; (vii) master attacker sends commands to run attack traffic on every assigned bot.

The scale of these experiments was the largest we could obtain in our testbed environment while guaranteeing the quality and reliability of the results. The testbed was constructed to be less dependent on the scale size with the objective to observe the distributed interaction between resolvers and the responsiveness of the defense. Note that the fraction of the attack's rate and the threshold is considered in the performance measurements. The performance metrics used in the experiments are: hit peak attack (HPA), DNS attack traffic over time, mitigation time, and system overhead.

3.3.3.1.3 Partial Deployment. DRS-ADAM is designed to propagate the requisite information sequentially from one resolver to another; i.e., by forming a chain of resolvers to reduce the message exchange complexity. If DRS-ADAM is not deployed on any one of the resolvers involved in an Amplified DNS Distributed Denial of Service Attack (ADD) attack, then it will break that chain of information propagation and form disjointed groups of resolvers that cannot exchange query rates with other groups. To avoid that situation, we enhanced our original victim and DNS resolver communication protocol to allow resolvers to receive multiple DNS resolver IPs from the victim monitor module based on the value k. This flexibility in k helps predecessor resolvers to be connected with successor resolvers when the chain is broken by a resolver that is not equipped with DRS-ADAM.

4.0 RESULTS AND DISCUSSION

The main outcomes of our research efforts can best be described as grouped into three main focal results. We therefore present and discuss these results separately in this section.

4.1 Performance of the DIAMoND Algorithm with Realistic Network Traffic on a Simplified Network Topology

4.1.1 Criteria for Evaluation of Detection. We consider three metrics: sensitivity, specificity, and overall accuracy. Sensitivity measures the proportion of malicious packets that are correctly identified as such, and specificity measures the proportion of legitimate packets that are correctly identified as such, whereas accuracy measures the proportion of packets correctly identified malicious and legitimate to all the packets.

Also, we quantify the additional information that is gained by deploying our system on top of independent surveillance agents. In other words, we ask by how much, if at all, the inclusion of the DIAMoND collaboration among nodes improves their accuracy relative to their use of only the local detection algorithms in isolation. We define information gain we using Kullback-Leibler (K-L) divergence. (Note: the potential for improvement in accuracy is scaled by the percent of malicious packets. Since in the case of network-wide stealth scans malicious packets constitute a smaller percentage of all network traffic, the increase in accuracy is strictly bounded; e.g. 0.045 represents a substantial improvement relative to the range possible for improvement.)

We explored the outcomes while varying the surveillance neighborhood size (TTL=1 or 2) for the communication overlay network.

4.1.2 Detection Performance.

Figure 10 depicts sensitivity as a function of (1 – specificity) for stealth scan attacks on an extended star network topology and an overlay network where neighborhoods are created on the basis of direct physical connections (TTL=1). The four experiments presented reflect the impact of four levels of importance (or trust) being assigned to the concern levels of neighbors. Results show a great improvement in sensitivity (between approximately 10% and 20% for low, med, and med+, high algorithms, respectively), without compromising specificity in comparison to purely parallel surveillance systems, operating independently. The reason why (1 – specificity) does not exceed 3:5% (in worst case) comes from two reasons: (i) precise calibration of the rate limiting sensitivity thresholds (i.e., the consensus of level of concerns of neighbors cannot reduce the sensitivity threshold of a chosen node below $st_{i,min}$), and (ii) level of concern of a node signals the anomaly, while the decision about the assigning particular flows to legitimate or malicious classes remain with independent detection. Lastly, the overall information gain of DIAMoND calculated over the accuracy of independent surveillance networks is approximately twice as large for med+ and high test functions as for low and med (between 0:022 and 0:047, cf. Table 2).

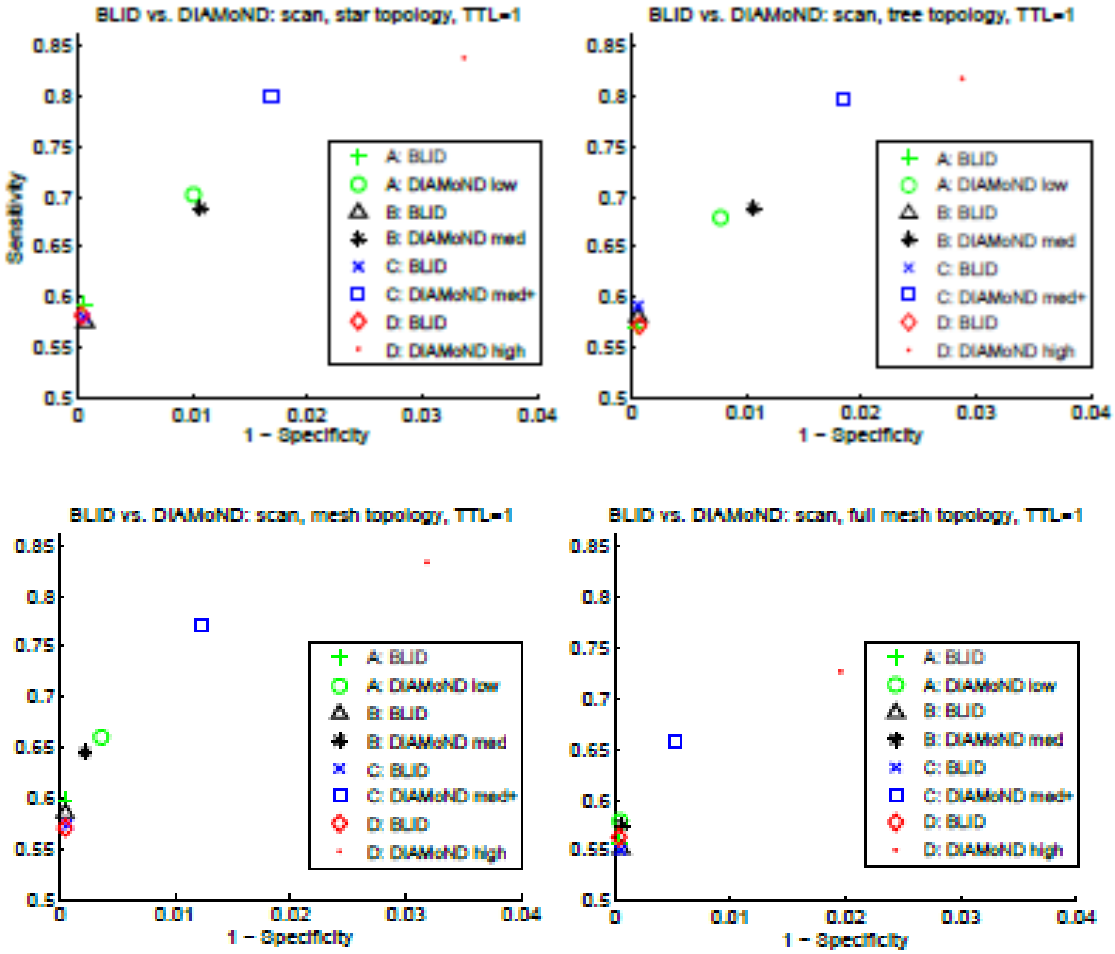

Figure 10. ROC Diagrams: Sensitivity as a function of 1–specificity. Comparison of DIAMoND and BLID for stealth scans in the TTL=1 neighborhood for different test functions (low, med, med+, and high), and for different topologies: star (top left), tree (top right), mesh (bottom left), and full mesh (bottom right).

Table 2. Sensitivity, 1–Specificity, Accuracy of BLID and DIAMoND

Sensitivity, 1–specificity (95% confidence interval), accuracy, and accuracy gain of DIAMoND over BLID for four test functions (A: low, B: med, c: med+, d: high).

	Sensitivity		1 - Specificity		Accuracy		
Test	BLID	DIAMoND	BLID	DIAMoND	BLID	DIAMoND	Gain
Stealth scan, star topology, TTL=1 neighborhood							
A	$0.594(\pm 0.024)$	$0.702(\pm 0.025)$	$6.4e^{-4}(\pm 2.2e^{-4})$	$0.01(\pm 0.002)$	0.893	0.914	0.022
B	$0.575(\pm 0.023)$	$0.688(\pm 0.023)$	$7.6e^{-4}(\pm 2,6e^{-4})$	$0.011(\pm 0.002)$	0.889	0.911	0.022
C	$0.58(\pm 0.02)$	$0.8(\pm 0.015)$	$6.2e^{-4}(\pm 1,5e^{-4})$	$0.017(\pm 0.003)$	0.889	0.935	0.047
D	$0.583(\pm 0.022)$	$0.837(\pm 0.013)$	$4.8e^{-4}(\pm 6.8e^{-5})$	$0.034(\pm 0.003)$	0.887	0.932	0.045
Stealth scan, tree topology, TTL=1 neighborhood							
A	$0.57(\pm 0.022)$	$0.679(\pm 0.02)$	$5.7e^{-4}(\pm 1.3e^{-4})$	$0.008(\pm 0.002)$	0.893	0.915	0.021
B	$0.581(\pm 0.021)$	$0.688(\pm 0.02)$	$6.7e^{-4}(\pm 1.9e^{-4})$	$0.011(\pm 0.002)$	0.891	0.911	0.021
C	$0.592(\pm 0.027)$	$0.797(\pm 0.028)$	$6.7e^{-4}(\pm 2e^{-4})$	$0.018(\pm 0.003)$	0.891	0.932	0.042
D	$0.571(\pm 0.025)$	$0.817(\pm 0.024)$	$7.9e^{-4}(\pm 3.8e^{-4})$	$0.029(\pm 0.003)$	0.89	0.932	0.042
Stealth scan, mesh topology, TTL=1 neighborhood							
A	$0.599(\pm 0.026)$	$0.66(\pm 0.028)$	$5.6e^{-4}(\pm 1.5e^{-4})$	$0.004(\pm 0.001)$	0.889	0.904	0.015
B	$0.586(\pm 0.022)$	$0.645(\pm 0.024)$	$5.5e^{-4}(\pm 1.4e^{-4})$	$0.002(\pm 6.6e^{-4})$	0.893	0.907	0.014
C	$0.574(\pm 0.019)$	$0.771(\pm 0.016)$	$7.2e^{-4}(\pm 2.2e^{-4})$	$0.012(\pm 0.003)$	0.89	0.932	0.043
D	$0.571(\pm 0.02)$	$0.833(\pm 0.018)$	$5.5e^{-4}(\pm 1.5e^{-4})$	$0.032(\pm 0.003)$	0.889	0.939	0.045
Stealth scan, full mesh topology, TTL=1 neighborhood							
A	$0.557(\pm 0.014)$	$0.578(\pm 0.016)$	$4.8e^{-4}(\pm 1.1e^{-4})$	$5.2e^{-4}(\pm 1.3e^{-4})$	0.881	0.887	0.006
B	$0.556(\pm 0.019)$	$0.574(\pm 0.021)$	$6.9e^{-4}(\pm 2.2e^{-4})$	$7.5e^{-4}(\pm 2.5e^{-4})$	0.884	0.89	0.006
C	$0.551(\pm 0.015)$	$0.657(\pm 0.023)$	$5.7e^{-4}(\pm 2e^{-4})$	$0.005(\pm 0.002)$	0.884	0.908	0.024
D	$0.562(\pm 0.016)$	$0.725(\pm 0.029)$	$5e^{-4}(\pm 1.1e^{-4})$	$0.02(\pm 0.004)$	0.882	0.912	0.03
DDoS attack, star topology, TTL=1 neighborhood							
A	$0.922(\pm 0.023)$	$0.938(\pm 0.02)$	$0.004(\pm 6.6e^{-4})$	$0.018(\pm 0.002)$	0.95	0.955	0.004
B	$0.92(\pm 0.016)$	$0.937(\pm 0.016)$	$0.004(\pm 6.5e^{-4})$	$0.018(\pm 0.003)$	0.95	0.955	0.005
C	$0.923(\pm 0.012)$	$0.962(\pm 0.01)$	$0.005(\pm 7.3e^{-4})$	$0.032(\pm 0.004)$	0.95	0.964	0.014
D	$0.922(\pm 0.03)$	$0.967(\pm 0.024)$	$0.004(\pm 7e^{-4})$	$0.038(\pm 0.004)$	0.95	0.965	0.014
DDoS attack, full mesh topology, TTL=1 neighborhood							
A	$0.922(\pm 0.023)$	$0.932(\pm 0.023)$	$0.003(\pm 6.2e^{-4})$	$0.008(\pm 0.001)$	0.949	0.953	0.005
B	$0.922(\pm 0.015)$	$0.931(\pm 0.015)$	$0.004(\pm 5.9e^{-4})$	$0.007(\pm 0.001)$	0.95	0.954	0.004
C	$0.921(\pm 0.018)$	$0.951(\pm 0.018)$	$0.004(\pm 4.8e^{-4})$	$0.017(\pm 0.002)$	0.949	0.963	0.014
D	$0.922(\pm 0.019)$	$0.959(\pm 0.018)$	$0.004(\pm 8e^{-4})$	$0.028(\pm 0.005)$	0.949	0.963	0.015
Stealth scan, star topology, TTL=2 neighborhood							
A	$0.575(\pm 0.029)$	$0.684(\pm 0.032)$	$6e^{-4}(\pm 1.3e^{-4})$	$0.008(\pm 0.002)$	0.89	0.912	0.022
B	$0.565(\pm 0.022)$	$0.676(\pm 0.024)$	$6.4e^{-4}(\pm 1.3e^{-4})$	$0.01(\pm 0.002)$	0.889	0.91	0.022
C	$0.557(\pm 0.021)$	$0.787(\pm 0.021)$	$7.5e^{-4}(\pm 5.2e^{-4})$	$0.019(\pm 0.003)$	0.889	0.932	0.045
D	$0.561(\pm 0.025)$	$0.843(\pm 0.014)$	$5.3e^{-4}(\pm 1.4e^{-4})$	$0.032(\pm 0.003)$	0.887	0.936	0.05
Stealth scan, star topology, TTL=3 neighborhood							
A	$0.599(\pm 0.027)$	$0.701(\pm 0.028)$	$7.3e^{-4}(\pm 1.7e^{-4})$	$0.009(\pm 0.002)$	0.887	0.909	0.023
B	$0.584(\pm 0.024)$	$0.68(\pm 0.026)$	$8.4e^{-4}(\pm 3.1e^{-4})$	$0.01(\pm 0.003)$	0.89	0.908	0.018
C	$0.568(\pm 0.029)$	$0.793(\pm 0.029)$	$6.1e^{-4}(\pm 1.7e^{-4})$	$0.02(\pm 0.003)$	0.887	0.932	0.045
D	$0.578(\pm 0.025)$	$0.851(\pm 0.015)$	$6.7e^{-4}(\pm 1.6e^{-4})$	$0.036(\pm 0.003)$	0.889	0.935	0.047
Stealth scan, star topology, attack correlation neighborhood							
A	$0.535(\pm 0.022)$	$0.658(\pm 0.025)$	$6.69e^{-4}(\pm 1.7e^{-4})$	$0.013(\pm 0.002)$	0.889	0.91	0.021
B	$0.563(\pm 0.02)$	$0.702(\pm 0.022)$	$9.57e^{-4}(\pm 5.8e^{-4})$	$0.016(\pm 0.003)$	0.888	0.914	0.027
C	$0.528(\pm 0.027)$	$0.752(\pm 0.027)$	$5.55e^{-4}(\pm 1.3e^{-4})$	$0.02(\pm 0.003)$	0.891	0.931	0.041
D	$0.531(\pm 0.023)$	$0.787(\pm 0.022)$	$5.8e^{-4}(\pm 1.6e^{-4})$	$0.04(\pm 0.004)$	0.891	0.924	0.034
Stealth scan, full mesh topology, attack correlation neighborhood							
A	$0.533(\pm 0.017)$	$0.586(\pm 0.021)$	$4.5e^{-4}(\pm 9.9e^{-5})$	$0.007(\pm 0.001)$	0.886	0.895	0.01
B	$0.528(\pm 0.015)$	$0.595(\pm 0.018)$	$8.2e^{-4}(\pm 3.4e^{-4})$	$0.01(\pm 0.002)$	0.887	0.896	0.01
C	$0.522(\pm 0.015)$	$0.672(\pm 0.017)$	$5.7e^{-4}(\pm 2.6e^{-4})$	$0.01(\pm 0.002)$	0.881	0.911	0.031
D	$0.532(\pm 0.014)$	$0.739(\pm 0.012)$	$5.4e^{-4}(\pm 1.1e^{-4})$	$0.03(\pm 0.002)$	0.886	0.913	0.027

4.1.3 Impact of Simplified Network Topologies. If there are no transit nodes, and we assume a full mesh1 topology, then the detection accuracy is decreased significantly in comparison to other topologies (cf. Figure 10 and Table 2), especially in case of the low and med excitation functions. This is because stealth scans are not as aggressive as DDoS attacks or worm propagation, so the great majority of nodes within a TTL=1 neighborhood forming a full mesh topology will not report any suspicious behavior.

Our results also indicate that, for aggressive DDoS attacks, the type of topology does not influence the accuracy (see Table 2). We observed that the information gain of the overlay detection system is lower (though always positive) in comparison with low-rate malicious activity, but the system can react close to the source of the attack more effectively and thereby reduce the collateral damage to minimum.

4.1.4 Minimal and Marginal Deployment Gain. Deployment of networked services across administrative boundaries usually has to take place progressively. We therefore also studied how deployment percentages affect performance of a DIAMoND system. In particular, we tried to understand the minimal deployment percentage needed for DIAMoND to have significant performance impact and marginal performance gain with additional deployment. To quantitatively evaluate deployment gain, we adapted a calculation of "offline marginal utility" originally proposed to analyze the impact of additional metrics [11].

Figure 11 shows a clear point of diminishing return: after 30% of the nodes participate in the DIAMoND system, the information gain is close to that achieved when all nodes are participating and the marginal deployment gain from increasing participation is insignificant. Conversely, even when only 10% of nodes are participating, the information gain is already over 0.01. When 20% nodes are participating, the information gain reached a significant 0.03. We thus concluded that, in this case: (i) minimal effective deployment is 10% of the network nodes participating, (ii) marginal gain is maximized at 20% deployment, and (iii) DIAMoND plateaus after 30% deployment, with minimal value gained by having additional nodes participating.

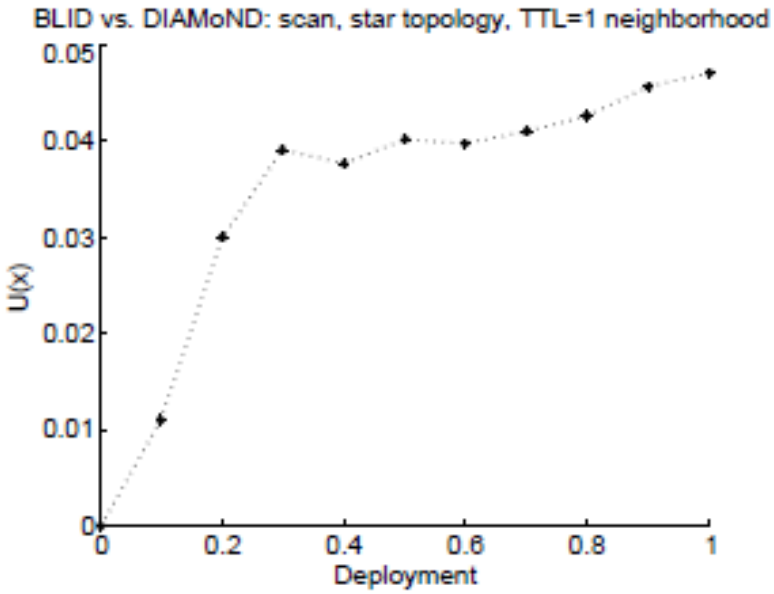

Figure 11. Information Gain of DIAMoND over BLID

Information gain as a function of the percentage of system deployment for scan activity in the TTL=1 neighborhood and the star topology.

4.2 Performance of the DIAMoND Algorithm as a DNS-Server Level Attack Detection and Mitigation System

4.2.1 Effectiveness at Controlling Hit Peak Attack Rate. Figure 12 shows the HPA metric with a varying number of name servers (nsIn) for different threshold settings; demonstrating the impact of an ADD attack on a victim with DRS-ADAM system in place. The peak attack traffic at the victim fluctuates with just small variations around the threshold absolute value (HPA is close to the threshold percentage) as we increase the number of resolvers; this is clear after 50 resolvers. As a result, inbound attack traffic at the victim will never reach its sustainable bandwidth. This implies that DRS-ADAM is scalable with respect to resolvers involved in attacks and suitable for mitigating large DNS amplification attacks without disrupting a victim's normal services.

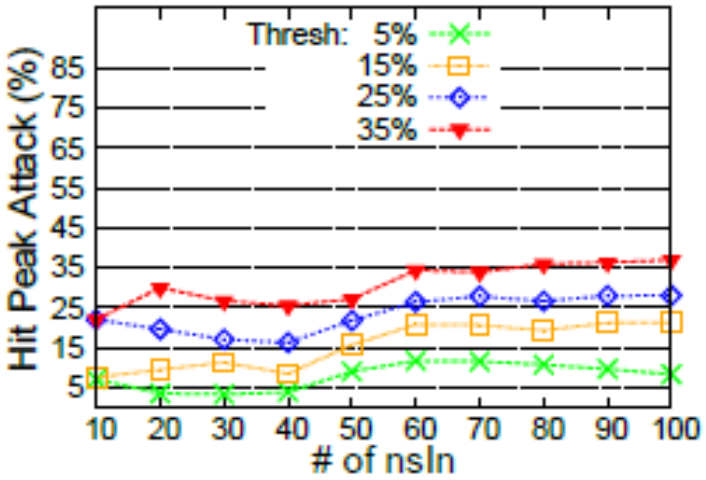

Figure 12. HPA Variation with the Number of Involved Name Servers

4.2.2 Mitigation Time. DRS-ADAM is a quick response system that can mitigate ADD attacks within a few seconds (see Figure 13). We expect even shorter mitigation times in the real-world, as the threshold will be much smaller while the attack growth speed is the same as in the test experiments.

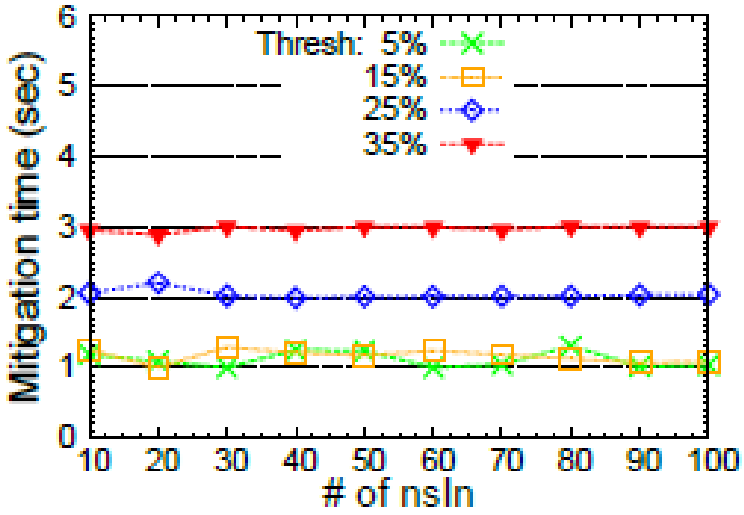

Figure 13. Mitigation Time as a Function of the Number of Involved Name Servers

Also, mitigation time does not vary with respect to the number of resolvers for a fixed threshold, which demonstrates the scalability of DRS-ADAM for mitigating real-world attacks. When DRS-ADAM is used, the victim's inbound attack traffic grows in seconds under an ADD attack, hits a peak value close to the threshold absolute value, and then declines substantially after

this highest point. The peak is short-lived, measured in ms, and the decline occurs within a few seconds (< 1:25s for any threshold and number of resolvers).

4.2.3 Bandwidth and Memory Consumption. A victim's outbound bandwidth consumption increases linearly in bytes per second with respect to number of resolvers involved in an ADD attack, as shown in Figure 14 (left). During an attack, a range of tens of thousands of resolvers corresponds to victim host bandwidth consumption of a few KBps. This consumption is negligible in comparison with the MBps or GBps bandwidth available to a victim, and thus, does not affect its system workload in any way. Similarly, the average bandwidth consumption for each resolver increases linearly with the number of resolvers involved in an attack, as shown in Figure 14(center). If we again scale the number of resolvers to tens of thousands, the worst case corresponds to a few hundred KBps bandwidth consumption per resolver. This amount of consumption is manageable and does not disrupt the main activities of a DNS resolver. Figure 14 (right) represents the total bandwidth consumption per resolver during the whole attack period.

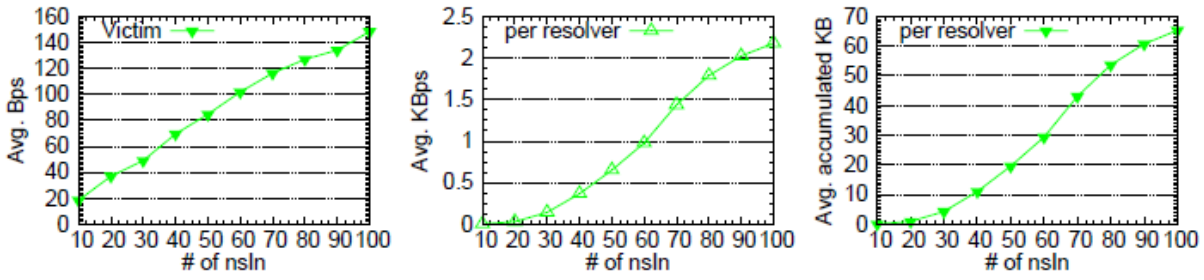

Figure 14. System Workload

4.2.4 Comparison with RRL. To show mitigation capabilities against highly distributed ADD attacks, this comparative experiment with RRL used a fixed attack size regardless of the number of resolvers. The absolute threshold value stayed the same for any threshold percentage, while the number of resolvers increased. As a result, the attack query rate per resolver decreased. To the best of our knowledge, this work presents the first experimental evaluation of BIND9's RRL feature in a distributed setting with multiple DNS resolvers; to date, RRL has been tested only on a single DNS server. Figure 15 depicts the normalized inbound rate (NIR) observed at the victim's router interface for different numbers of resolvers. The threshold used is 5%. NIR is the real-time attack rate percentage of the peak expected attack's volume at the victim (the total attack volume after the growth phase when no mitigation is in place). The responses-per-second value of RRL is set to 10 based on the 5% threshold. When the number of resolvers is greater than or equal to 20, the RRL rate-limit response is not activated during the attack, and the victim host faces downtime. When the ADD attack becomes well-distributed, using more resolvers at the same attack size, it can bypass the RRL detection mechanism when the attack query rate per resolver becomes less than or equal to the per-second limit of RRL (10 qps in the 20 resolvers' case). Thus, RRL will have little or no impact on highly-distributed ADD attacks. Even when RRL can detect an attack in the case of 10 resolvers, it cannot stop the attack traffic before the HPA hits 34:8% for the 5%

threshold value. This trait shows that high-scale attacks might still be effective even if they are detected and rate-limited. In contrast, DRS-ADAM stops all attack traffic in less than two seconds and contains HPA at around 5%.

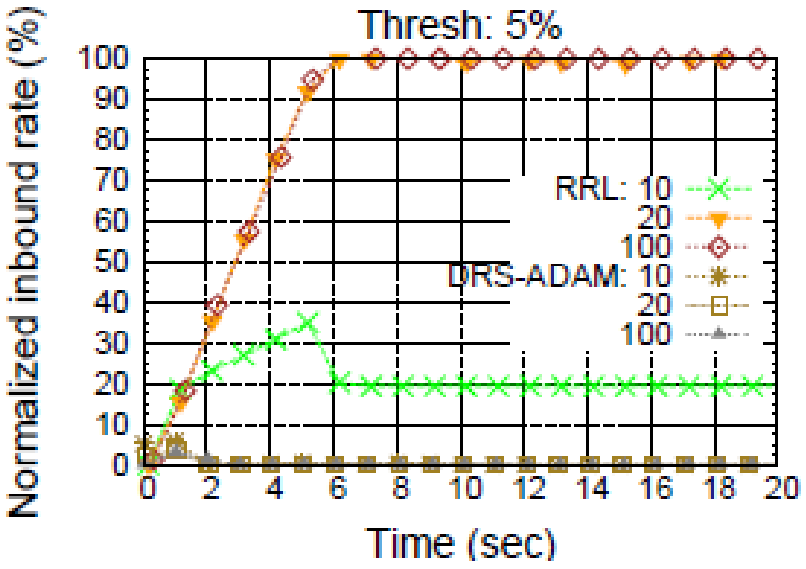

Figure 15. Normalized Inbound Rate over Time of Victim's Router

4.2.5 Partial Deployment of DRS-ADAM. Figure 16 shows HPA for different percentages of resolvers equipped with DRS-ADAM. Resolvers that are equipped with DRS-ADAM were randomly selected from a set of 100 resolvers. In particular deployment scenarios, there will always be a minimum HPA value, since resolvers that are not equipped with DRS-ADAM cannot stop an ADD attack. The best performance line in the graph represents this minimum HPA that will be observed when a theoretically best-possible, resolver-based (both local and distributed) solution is deployed. By increasing k, the DRS-ADAM performance improved, and eventually when $k \geq 5$, gave close to that best theoretical performance. The message complexity for the victim would now be O(kn), increasing linearly with the value of k. For a small constant k value; however, it is safe to expect the complexity to remain at O(n).

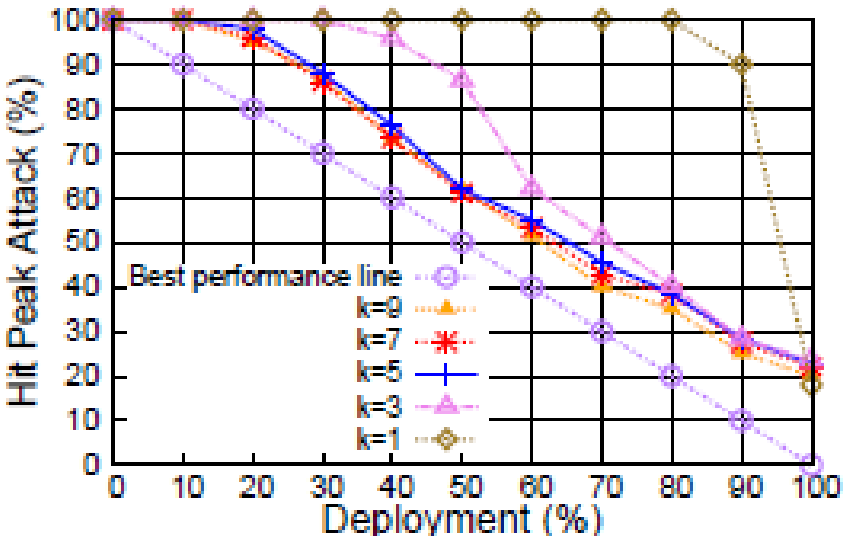

Figure 16. HPA Variation for Partial Deployment of DRS-ADAM
Deployment with 100 resolvers and 15% Threshold.

4.2.6 Manipulating DRS-ADAM and Defending Against UDP Spoofing. The attacker can use various methods to manipulate (or game) our DRS-ADAM system by exploiting its communication protocol. In one scenario, an attacker can try to falsify the DNS query rates or unknown-list circulating among DNS resolvers by forging UDP packets. The attacker can send a resolver packet of less than the actual DNS query rate (even 0). DRS-ADAM solves this problem by considering the packet with the maximum DNS query rate but limited to an upper rate bound. This received attack rate limit prevents a bogus message with a large rate from blocking normal DNS traffic to a client. An attacker can also send a huge garbage unknown-list to increase a DNS resolver's communication overhead. DRS-ADAM cross validates list by obtaining the DNS query rate for a few random resolvers; the rate should be greater than 0. We implemented both these logics in our DRS-ADAM system and simulated a manipulation attack by constantly generating forged UDP packets (with an interval of 100 ms) that each carried either a lower DNS query rate or a huge garbage list. Our DRS-ADAM successfully dropped all such packets and our results remained same as shown in Figures 12 and 13.

Another scenario exploits DRS-ADAM to block normal DNS traffic even when the victim is not under ADD attack. Here, the attacker sends forged packets of DNS query rates. DRS-ADAM cross validates query rates from randomly selected resolvers just before blocking the victim services. The attacker can further attempt confusion by sending responses from all resolvers. DRS-ADAM can identify the type of attack as it does not expect packets from all resolvers. For this experiment, we sent forged DNS query rates along with normal DNS queries from the victim to a resolver. Figure 17 shows that the DRS-ADAM manipulation prevention system successfully continued serving DNS queries for the victim even when an attacker tried to exploit it.

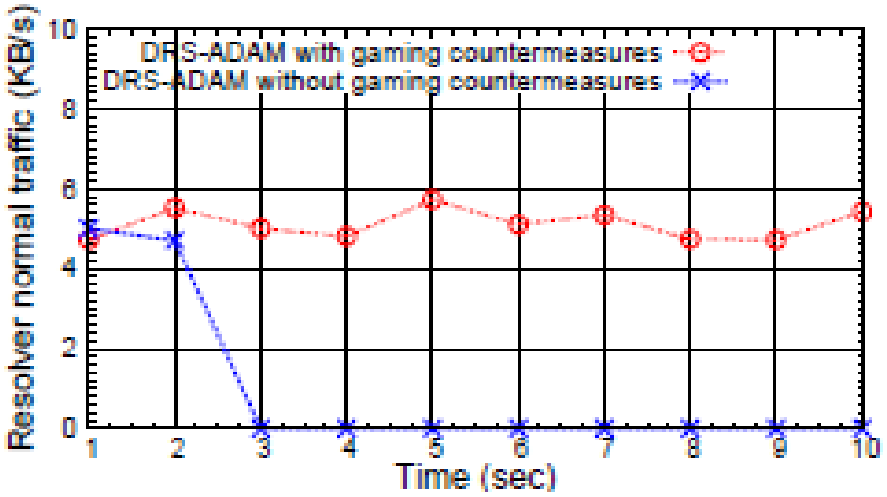

Figure 17. Manipulating DRS-ADAM and Observed Resolver Normal Traffic

4.3 Performance of the DIAMoND Algorithm with Simplified Network Traffic on more Realistic Network Topologies

4.3.1 Performance on Different Topologies Assuming Gaussian-Distributed Traffic. Both DIAMoND and the independent parallel algorithm can in general detect the attack successfully when the normal underlying traffic is Gaussian, regardless of the underlying topology (Figure 18). On average, the accuracy in square lattices is higher than 90%, and in all other topologies it is around 80%. These values depend on the 'intensity' of the attack, i.e. the packet size, z, and the fraction of compromised nodes, p. (Note: these two parameters do not have the same effect on the results.) The detection algorithms are more efficient when a small number of nodes use large packet sizes, compared to when more nodes use smaller packets.

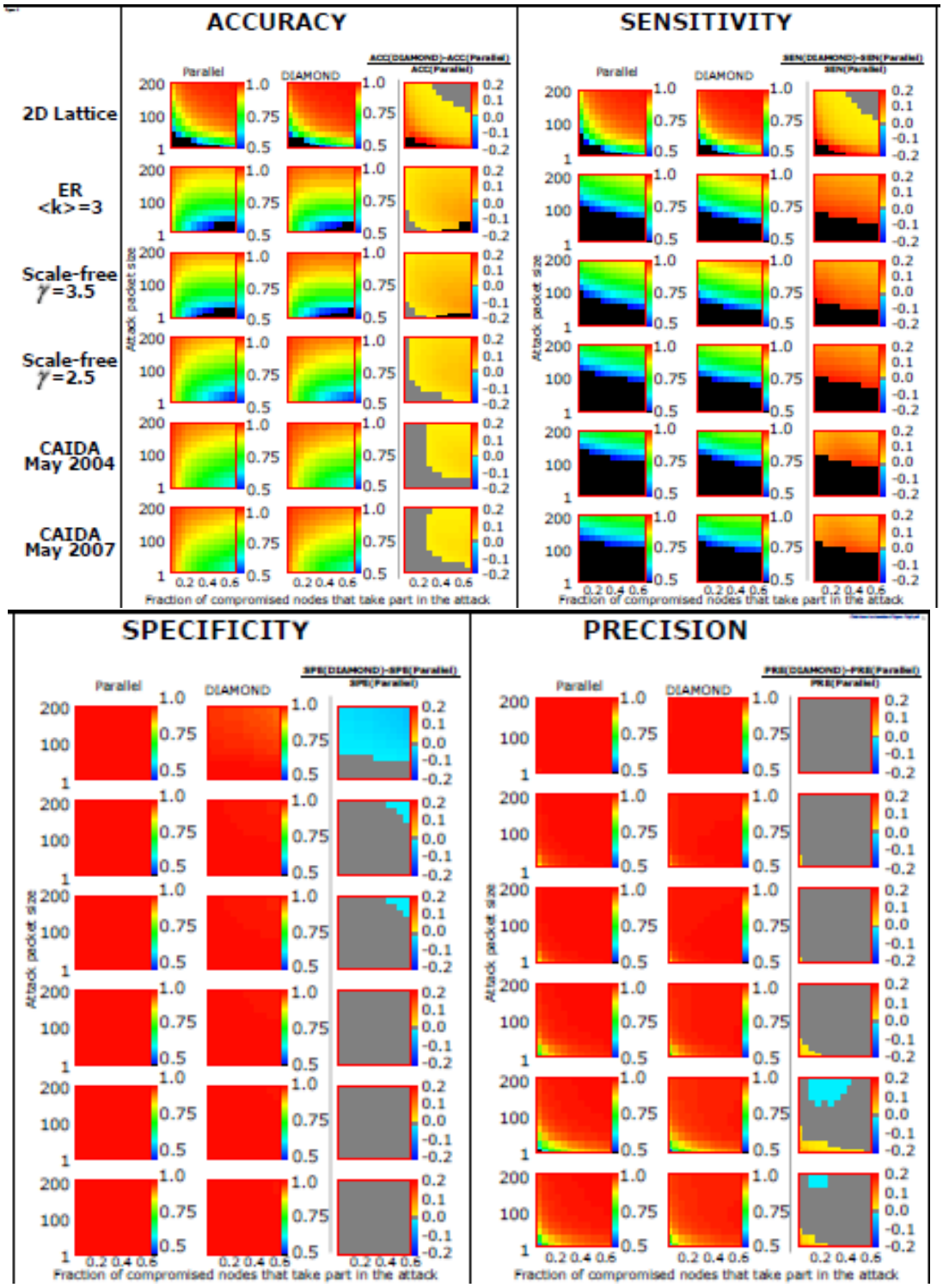

Figure 18. Accuracy, Sensitivity, Specificity, and Precision for Gaussian Traffic

Values of accuracy, sensitivity, specificity, and precision for Gaussian distributed traffic. For each index, the first column shows the results for the parallel algorithm, the second column shows the results for DIAMoND, and the third column displays the relative gain of the DIAMoND algorithm over the parallel algorithm. We use the upper threshold, si0, as the threshold for the parallel algorithm.

DIAMoND significantly outperforms the accuracy of purely parallel detection, in some cases as much as 20%. For the lattice structure, the largest improvement is in the regime of small packets, where attack detection is more difficult. In random model networks, ER and scale-free, there is an improvement of roughly 10%, which in the Internet structures becomes around 7%. For the Internet, the advantage of DIAMoND emerges when both z and p have large values. If one of the two is small, then both algorithms are equally efficient.

The improved performance of DIAMoND is mainly due to the successful detection of positive hits. In the sensitivity plot, DIAMoND detects a significantly larger fraction of attacked nodes compared to the parallel algorithm. The rate of detection is rather low, in both cases, when the packet size is small, independently of how many nodes take part in the attack. The detection seems to be more efficient in lattices, but in model and real networks the sensitivity increases to 80% only when the packet size is ~100 (i.e. roughly 10% of the average regular traffic). For values lower than that, neither algorithm can have sensitivity larger than 50%. In all cases, though, the shared information scheme results to a much improved detection rate of true positive hits, with a gain of 10-20%. The two algorithms yield comparable specificity results with each other. Specificity is always very close to 100%, which indicates that it is difficult to mistake regular traffic for attack traffic, and there are very few false positives. This explains also the behavior of the precision, since almost all hits detected as positive are indeed true positives. The main drawback of both algorithms is, therefore, that there are many false negatives, especially in the parallel algorithm. In summary, DIAMoND is efficient in separating true from false positives, but there are also many false negatives, i.e. undetected positive hits.

The above comparisons were made when the threshold for the parallel algorithm was equal to the upper threshold of DIAMoND, si0. We examined how this comparison changes when the parallel threshold takes the value of the lower threshold, siL. As shown in Figure 19, the accuracy of the parallel algorithm increases, and now it is higher than in DIAMoND. This increase is mainly due to the improvement of sensitivity, which is also larger than in DIAMoND. The specificity, though, which above was always close to 100% now suffers a large drop, and even becomes smaller than 50%. As a consequence, the measurement of precision also deteriorates a lot. This behavior can be explained because when we lower the threshold it becomes easier to detect attacks, but at the same time we falsely characterize legitimate traffic as attack. In the extreme case, a very low threshold can characterize all traffic as attack, with the tradeoff that it is no longer possible to identify regular traffic.

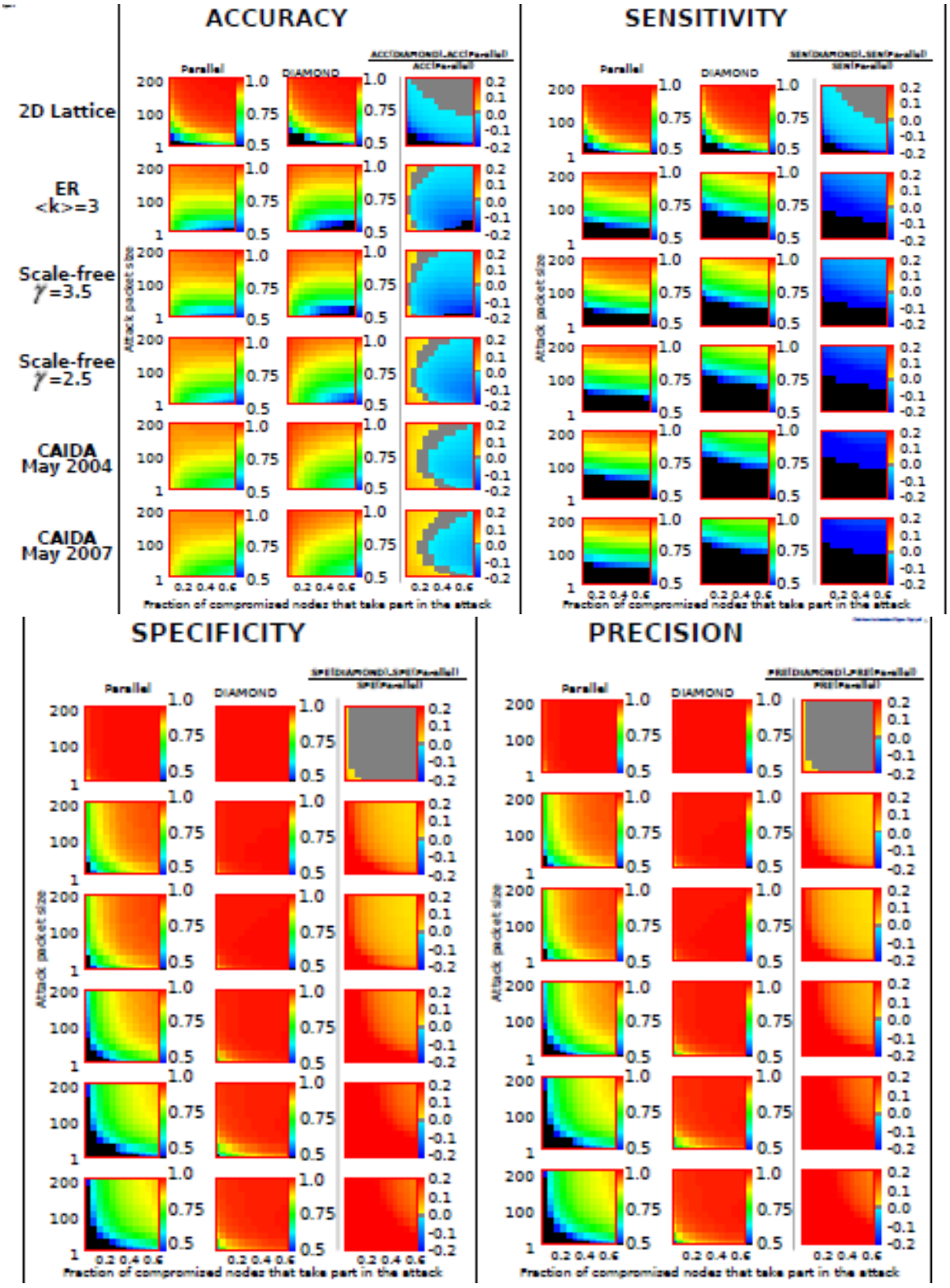

Figure 19. Additional Results for Gaussian Traffic

Same as Figure 18, but now for the parallel algorithm we use the lower threshold, siL.

4.3.2 Performance on Different Topologies Assuming Uniformly Distributed Traffic. In general, the accuracy improves significantly, both in DIAMoND and the no-sharing algorithms (Fig. 20). Except for very small values of the packet size z, accuracy is now larger than 90%. The network structure has a minor influence in accuracy and the results are largely agnostic to the substrate. All the measures that we calculated - accuracy, sensitivity, specificity, and precision – have values close to 100%. There is not a notable gain in using one algorithm over the other, but the precision and specificity seem to work better for the parallel algorithm. The gain is minimal, though, because both algorithms are very close to perfect detection for all values of z and q.

The increased detection efficiency, compared to Gaussian traffic in Figure 18, is due to the enhanced sensitivity. The algorithms can separate attack from noise better now, because they succeed in the detection of true positives, and there are very few false negatives.

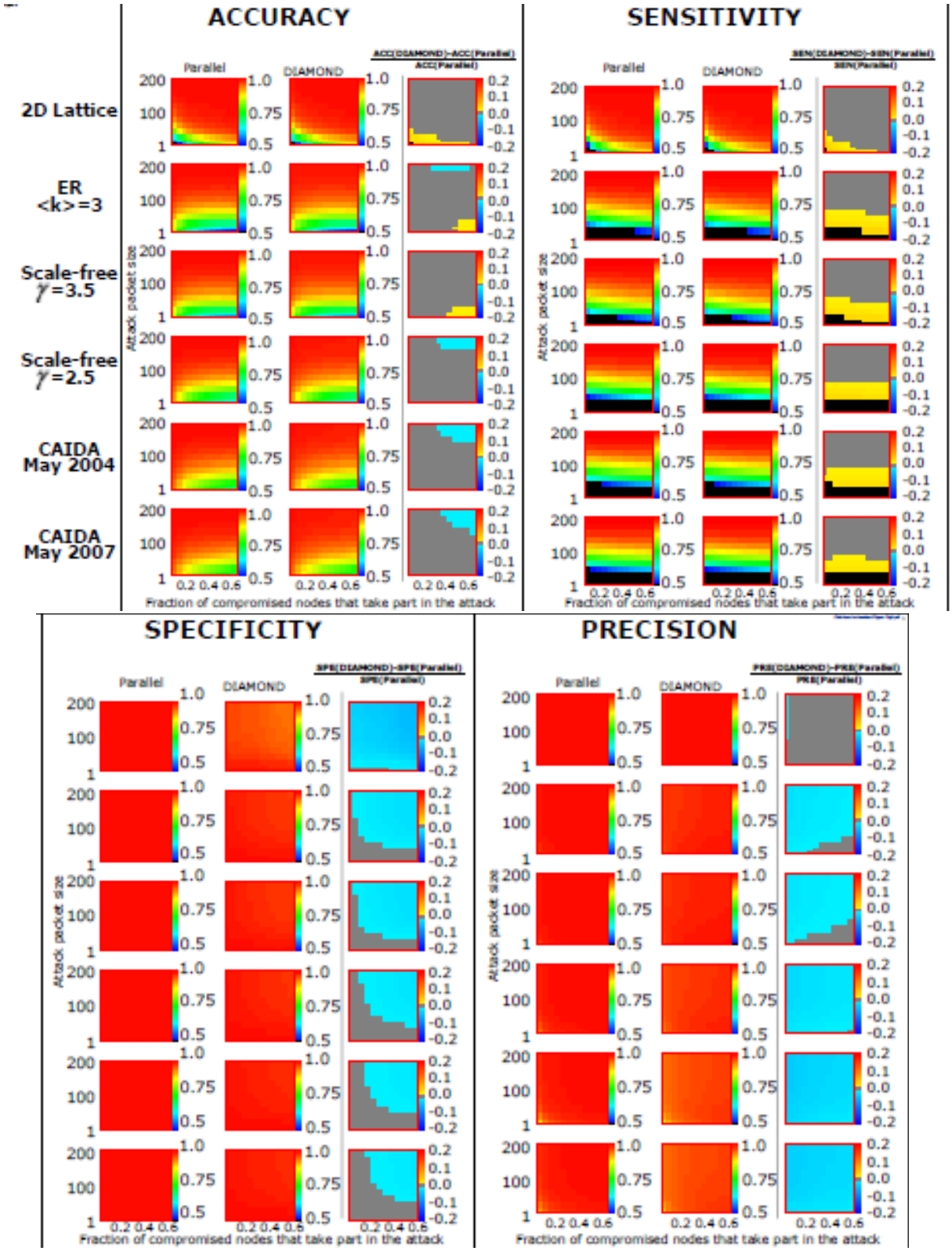

Figure 20. Accuracy, Sensitivity, Specificity, and Precision for Uniformly Distributed Traffic

For the parallel algorithm, we use the upper threshold, si0.

In the above discussion, the threshold for the parallel algorithm was fixed to the upper DIAMoND threshold value, si0, and there was no clear advantage of one method over the other. When we move the parallel threshold to the lower value, siL, then DIAMoND outperforms the parallel algorithm again. The results for the parallel algorithm become less accurate, and specificity decreases significantly. The metric that mainly suffers from the lower threshold is precision because now false positives increase. In cases of small packet size z and/or small fraction of compromised nodes, there is a dramatic change of precision from an almost perfect precision (for the upper threshold) to a complete failure (less than 50% for the lower threshold).

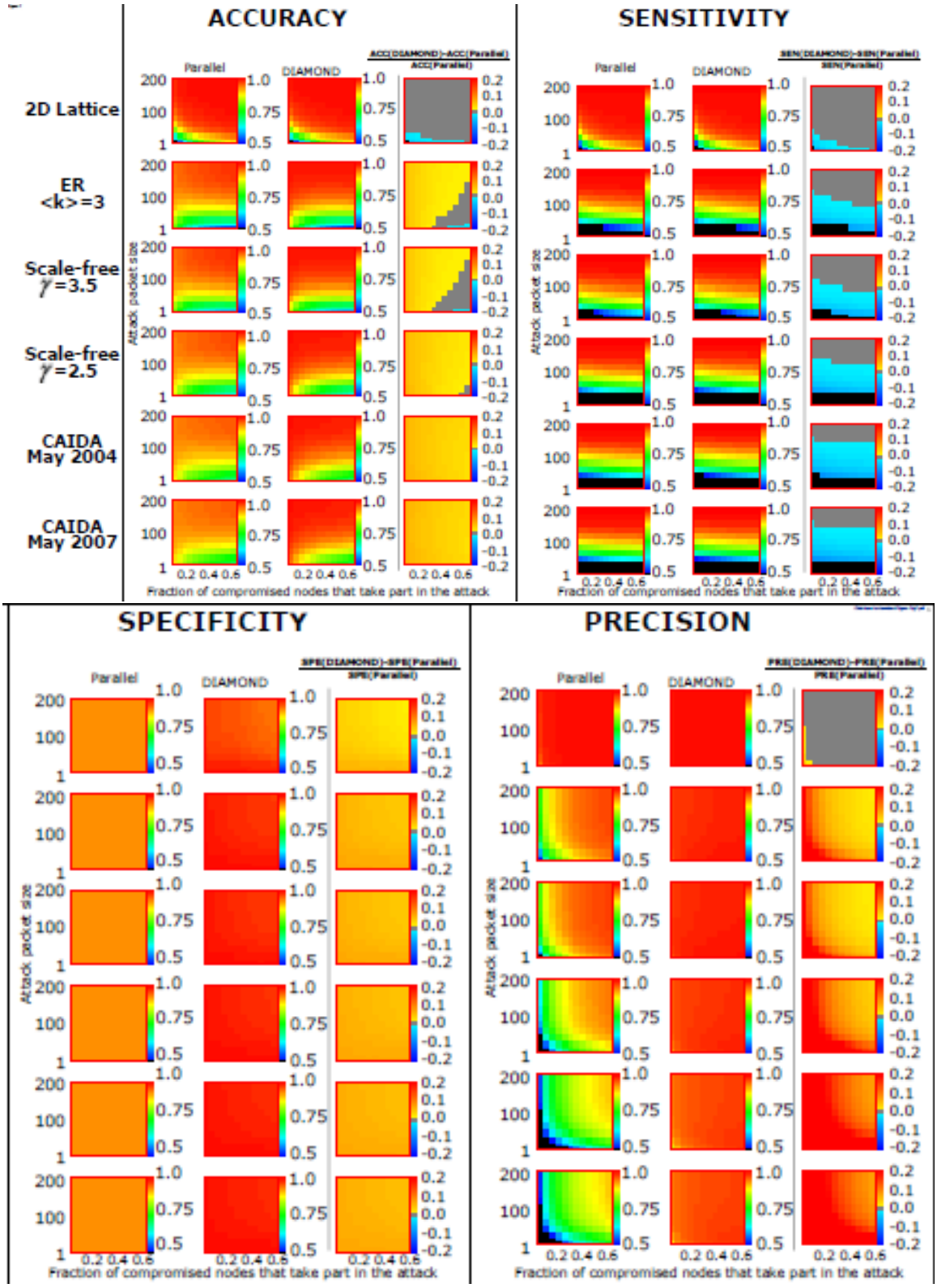

Figure 21. Additional Results for Uniform Traffic

Same as Figure 20, but now for the parallel algorithm we use the lower threshold, siL.

4.3.3 Performance on Different Topologies Assuming Exponentially Distributed Traffic. In this case, we do not expect the detection algorithms to be efficient, because even regular traffic can assume very large values. Therefore, without any additional information it is very hard to determine if increased traffic is due to an attack or not.

This behavior is verified in Fig. 22. The accuracy for lattices, ER networks, and (partially) scale-free networks with weak hubs is very low, and in most cases does not even exceed 50%. For systems with strong hubs, such as the real CAIDA networks that we studied, the accuracy increases, but still remains relatively low, at the level of 75%. The sensitivity fails to reach 50% under any circumstances, but the specificity is very high. This simply means that the algorithms cannot detect positive hits, but they do not have a problem in detecting the absence of attacks (true negative). The precision is quite high, which indicates that when a positive hit is detected, then there is a low probability that this is a false positive. In short, for exponential traffic, the algorithms are successful when they identify an attack, but they cannot identify the majority of attacks, leading to a huge number of false negatives.

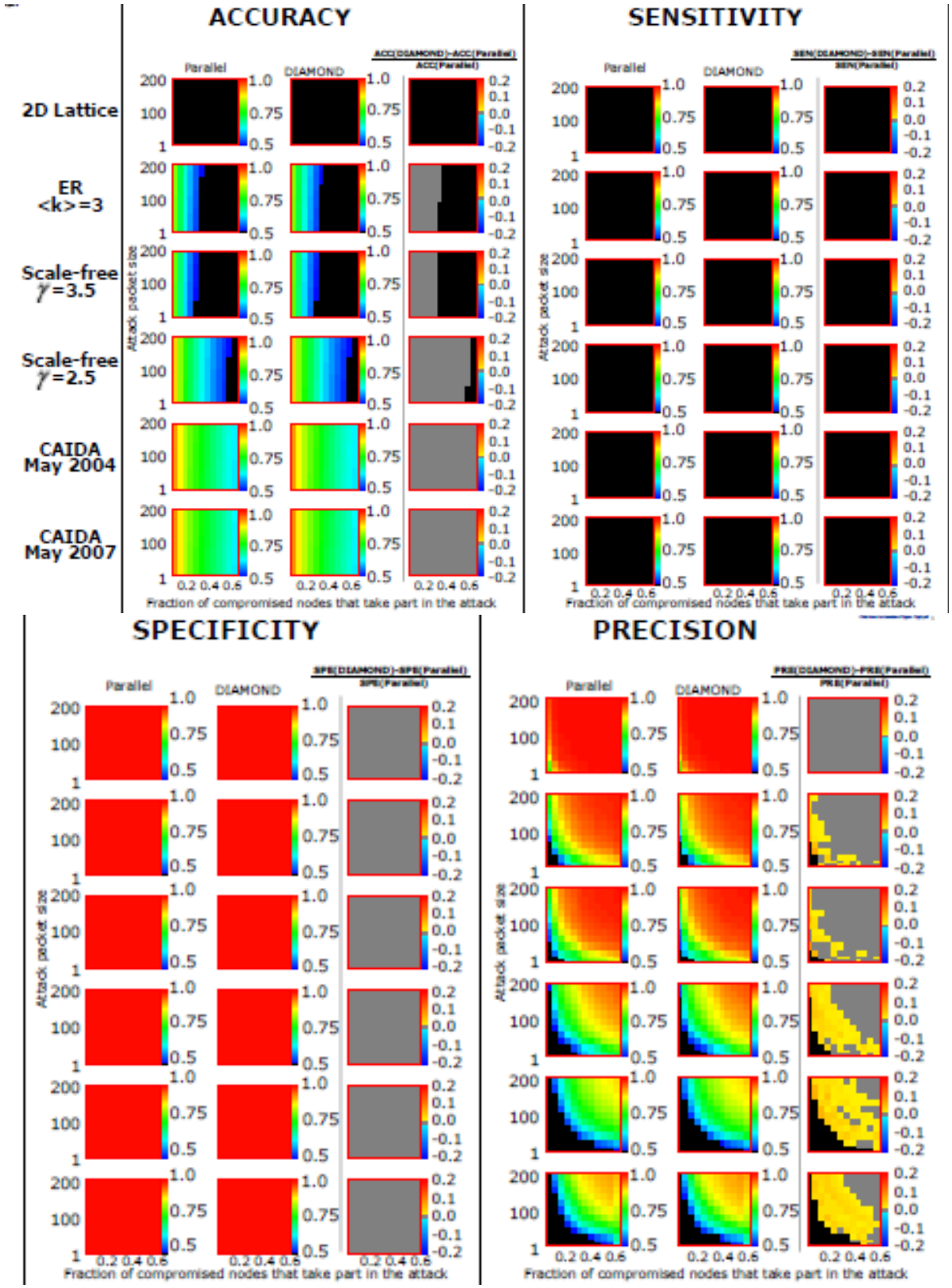

Figure 22. Accuracy, Sensitivity, Specificity, and Precision for Exponentially Distributed Traffic
For the parallel algorithm we use the upper threshold, si0.

The differences between the two algorithms are not significant, even though DIAMoND outperforms the parallel algorithm in precision. When we lower the threshold (Fig. 23), the results for the parallel algorithm deteriorate and the detection of true negatives drops by almost 10%. In these cases, DIAMoND offers an overall better performance in accuracy, specificity, and precision.

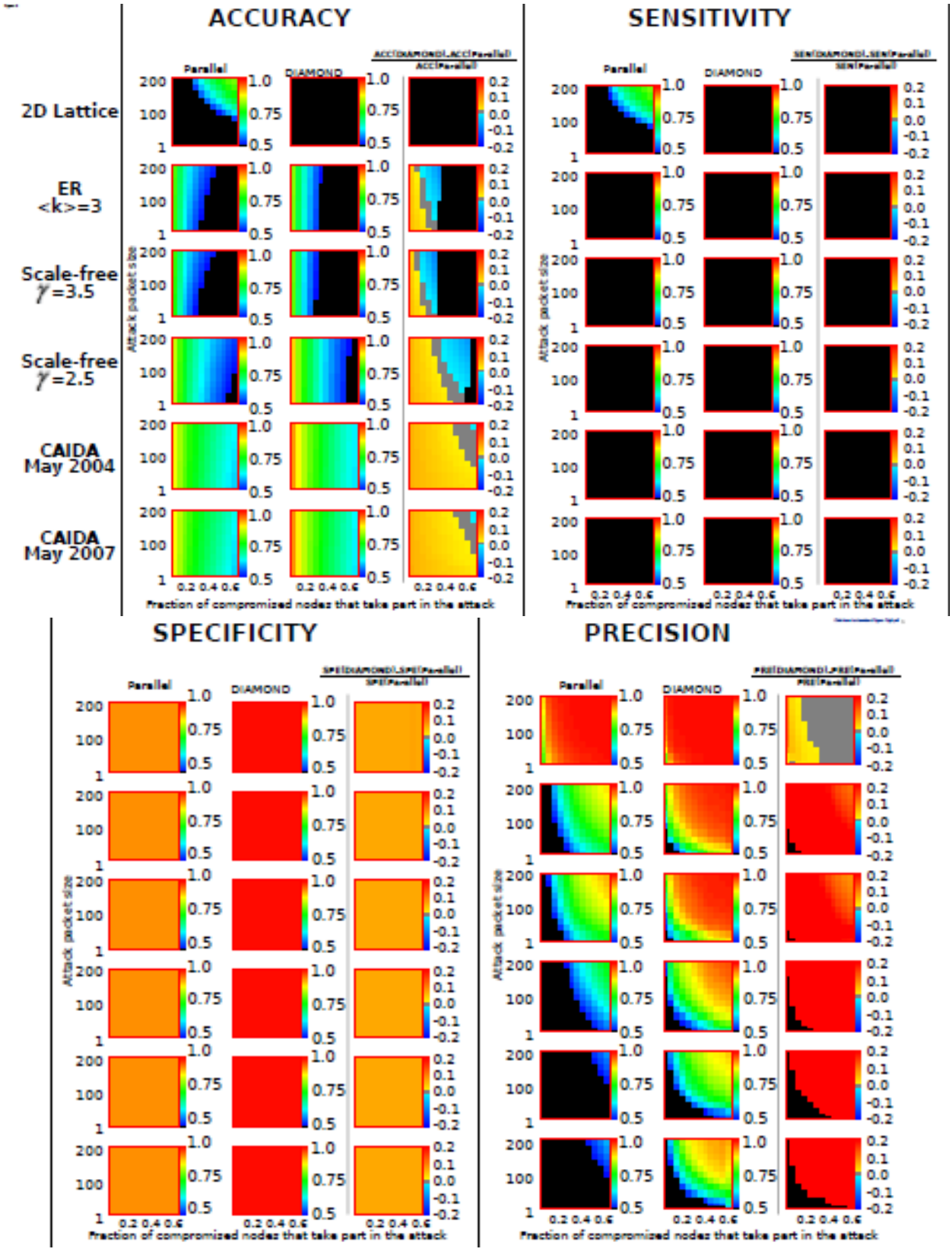

Figure 23. Additional Results for Exponential Traffic

Same as Figure 8, but now for the parallel algorithm we use the lower threshold, siL.

5.0 CONCLUSIONS

5.1 The Utility of the Bio-Inspired DIAMoND Algorithm as a Surveillance System Across Threat and Normal Network Traffic Scenarios

Based on algorithms observed in colonies of honey bees, our proposed self-organizing, nonparametric distributed coordination framework relies on dynamic individual detection thresholds for anomalous event pattern detection on networks. DIAMoND can therefore be used with any set of local anomaly detection schemes. We have already demonstrated DIAMoND's ability to improve accuracy in detection for network-wide stealthy port scan and SYNflooding-based DDoS directly on an emulation testbed. DIAMoND demonstrated up to 20 percent enhancement in sensitivity without sacrificing specificity. DIAMoND also proved robust in simulation to variation in neighborhood size, communication protocol, and partial deployment. By leveraging only nonparametric information sharing, resulting from independent local anomaly detection schemes, DIAMoND allows multiple entities, which may be functionally and/or legally prohibited from sharing cyber data, to leverage each other's insight and increase their effectiveness in cyber defense. Furthermore, DIAMoND enables real-time adaptation in anomaly defintion, eliminating the identification-designed-response delay inherent in defenses that react to known and predefined threats, and allowing active defense for emerging novel network attacks.

5.2 The Utility of DIAMoND as a DNS-Server Detection and Mitigation System

Existing mitigation techniques for ADD attacks cannot prevent a victim's network bandwidth from being fully consumed even with timely detection of the attack. The DIAMoND-based system, DRS-ADAM, can quickly detect ADD attacks and fully stop them well before they pose a significant threat to victims. To detect an ADD attack, DRS-ADAM utilizes the DNS resolvers that are exploited in the attack and does not require any additional, dedicated hardware. DNS resolvers share the local DNS query rates with each other, facilitating quick and accurate computation of accumulated query rates. Our simulation results show that DRS-ADAM is scalable for both mitigation times and hit peak attack rates. ADD attacks will always be contained close to victim's acceptable thresholds, regardless of the number of DNS resolvers involved in an attack, and mitigated within a few seconds. DRS-ADAM is robust against manipulation scenarios such as falsifying the DNS query rates shared among DNS resolvers (with UPD packet forging).

5.3 The Utility of DIAMoND Across Various Realistic Network Topology Scenarios

DIAMoND's performance was assessed for different traffic substrates, different distributions of regular traffic, different sizes of the attack packets, and different number of nodes taking part in the attack. The results of the algorithm were evaluated against a parallel algorithm, where there was no information sharing among the participating nodes. In the majority of the cases, DIAMoND outperformed the parallel algorithm by a wide margin. In a few cases, the opposite was actually true. Even in these cases, though, the information-sharing algorithm was an improvement in some aspect, e.g. performing better in precision even though accuracy was worse.

Sharing non-parametric conclusions (i.e. level of concern) with neighboring nodes allows for an efficient detection of DDoS attacks, and many times this is done with a near perfect accuracy. The comparison of this scheme with a strictly local algorithm lacking communication, shows that the DIAMoND is more reliable in the majority of the studied conditions.

5.4 Overall Conclusion

The research completed with the support of this funding has successfully designed a bio-inspired anomaly detection algorithm that improves upon performance of strictly parallel surveillance systems without incurring the costs of centralized analysis, and without requiring the trust and/or secure communication channels that would enable sharing of sensitive data for collaborative, hierarchical analysis. While further questions remain open for investigation and algorithmic refinement (including, but not limited to, the sensitivity of the system to greater diversity of hierarchical scale of DIAMoND system participants, the impact of multi-modal surveillance data, and the impact of colluding malicious participants on system performance), these outcomes provide a firm foundation for nonparametric collaborative surveillance systems. They have now been presented for peer review by academic researchers in conference presentations and proceedings[3, 12], magazine articles[13], journal articles [15], and book chapters [16].

6.0 REFERENCES

1. Chandola, V., Banerjee, A., and Kumar, V., "Anomaly detection: A survey." *ACM Computing Surveys (CSUR)*, **41**(3), 2009, pp. 15.
2. Gadau, J., Fewell, J., and Wilson, E.O., Organization of Insect Societies: From Genome to Sociocomplexity, Harvard University Press, Cambridge, MA, 2009.
3. Korczynski, M., Hamieh, A., Huh, J.H., Holm, H., Rajagopalan, S.R., and Fefferman, N.H., "DIAMoND: Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection," *The 24th International Conference on Computer Communications and Networks (ICCN)*, Las Vegas, Nevada, 2015.
4. Open Networking Lab, "OpenFlow," Pox Wiki, URL: https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-forwarding.l2_learning . Accessed November 1, 2015.
5. Korczynski, M., Janowski, L., and Duda, A., "An Accurate Sampling Scheme for Detecting SYN Flooding Attacks and Portscans,"*IEEE International Conference on Communications (ICC)*, Kyoto, Japan, 2011.
6. Erdos, P. and Rényi, A., "On the Evolution of Random Graphs," *Publ. Math. Inst. Hung. Acad. Sci*, **5**(1), 1960, pp. 17-60.
7. Barabási, A.-L. and Albert, R., "Emergence of scaling in random networks,"Science," **286**(5439), 1999, pp. 509-512.
8. Molloy, M. and Reed, B., "A Critical Point for Random Graphs with a Given Degree Sequence," *Random Structures & Algorithms*, **6**(2-3), 1995, pp. 161-180.
9. Leskovec, J., Kleinberg, J., and Faloutsos, C., "Graphs Over Time: Densification laws, Shrinking Diameters and Possible Explanations," *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, Chicago, IL, 2005.
10. Jose, A.S. and Binu, A., "Automatic Detection and Rectification of DNS Reflection Amplification Attacks with Hadoop MapReduce and Chukwa," *Fourth International Conference on Advances in Computing and Communications (ICACC)*, Cochin, India, 2014.
11. Barford, P., Bestavros, A., Byers, J., and Crovella, M., "On the Marginal Utility of Network Topology Measurements," *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, Burlingame, CA, 2001.
12. Verma, S., Hamieh, A., Huh, J.H., Holm, H., Rajagopalan, S.R., Korczynski, M., and Fefferman, N.H., "Stopping Amplified DNS DDoS Attacks Through Distributed Query Rate Sharing," *The 11th International Conference on Availability, Reliability and Security (ARES)*, Salzburg, Austria, 2016.
13. Korczynski, M., Hamieh, A., Huh, J.H., Holm, H., Rajagopalan, S.R., and Fefferman, N.H., "Hive Oversight for Network Intrusion Early Warning using DIAMoND: a Bee-Inspired Method for Fully Distributed Cyber Defense," *IEEE Communications Magazine*, **54**(6), 2016, pp. 60-67.
14. Korczynski, M. and Fefferman, N.H., "DIAMoND: Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection," URL: http://mkorczynski.com/diamond.html .
15. Gallos, L., Korczynski, M., and Fefferman, N.H., "Anomaly Detection Through Information Sharing Under Different Topologies," *EURASIP Journal on Information Security*, in press.
16. Korczynski, M., Hamieh, A., Huh, J.H., Holm, H., Rajagopalan, S.R., and Fefferman, N.H., "DIAMoND: Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection (invited extended version)," *Security, Privacy and Reliability in Computer Communications and Networks*, eds. Sha, K., Striegel, A., and Song, M., River Publishers Series in Communications, Gistrup, Denmark, 2017.

APPENDIX A – Full Description of Software Simulation Testbed

The following is quoted directly from a simulation instruction website we developed and maintain [14].

Getting Started

We have developed our prototype communication controller as an OpenFlow component in the POX environment and evaluated in Mininet 2.0 simulator. It has been actively developed and supported. We encourage you to contribute code, bug reports, and anything else that can improve the proposed controller.

The easiest way to get started is to download a pre-packaged Mininet VM that includes Mininet itself, all OpenFlow binaries and pre-installed tools. In addition to standard Mininet installation, we provide our POX controllers, prototype implementation of the communication protocol, and some testing scenarios. To install Mininet please download our Mininet VM image (based on Mininet version 2.0.0) and follow steps 2-5 of the Mininet VM Installation (recommended), as described here. After the installation, we encourage to do the walkthrough and run the OpenFlow tutorial.

DIAMoND Components in POX

In this section, we discuss the DIAMoND components in POX, whereas the description of standard components that come with POX can be found here.

l2_learning-modif This component makes OpenFlow switches act as a "standard" type of L2 learning switch analogically to forwarding.l2_learning. However, it ignores the distributed-detection communication packets (Ethernet type 0x0105) that are exchanged between switches participating in the DIAMoND network.

local_detectors This component can detect all kind of anomalies related to the TCP protocol locally, as described in the following paper: "An Accurate Sampling Scheme for Detecting SYN Flooding Attacks and Portscans".

anomaly_detector This component is the most important part of the DIAMoND algorithm. In first place it creates neighborhoods (smaller subnetworks within which the information is exchanged) based on hop limits. Second, it combines levels of concern of its neighbors with a report from its internal NIDS (the local_detectors component).

This component has two options to determine the scope of the neighborhood and the level of trust of a switch to its neighbors:

--ttl defines the neighborhood based on a hop limit that reflects the geographical or administrative distance. In other words, nodes exchange their levels of concerns with their direct neighbors (ttl=1) or their neighbors and neighbors of their neighbors (ttl=2), and so on.

--algo defines the level of trust to all switches belonging to node's neighborhood. Possible values are: "low", "med", "med+", and "high".

Example run command syntax

```
sudo ./pox.py anomaly_detector --ttl=2 --algo="med" l2_learning-modif
openflow.discovery log.level --WARNING --anomaly_detector=DEBUG
--local_detectors=DEBUG
```

And example run command syntax to build the Mininet network:

```
sudo mn --switch ovsk --controller=remote,ip=127.0.0.1,port=6633 --custom
~/mininet/custom/topoStar7pro.py --topo star7pro
```

anomaly_detector2 This component has one more option in comparison to anomaly_detector. --strategy defines the strategy of neighborhood creation. In addition, neighborhood based on the hop limit (option: "neig"), we propose to correlate previously observed attacks and construct neighborhoods based on the assumption that malicious activity may reoccur and be launched from the same set of compromised machines and/or against the same victims (option: "anom")

Example run command syntax

```
sudo ./pox.py anomaly_detector2 --ttl=1 --algo="med+" --strategy="anom"
l2_learning-modif openflow.discovery log.level –WARNING
--anomaly_detector2=DEBUG --local_detectors=DEBUG
```

You may also run the following command with physical Mininet topologies that contain loops:

```
sudo ./pox.py anomaly_detector --ttl=1 --algo="med" l2_learning-modif
openflow.discovery openflow.spanning_tree --no-flood --hold-down log.level
--WARNING --anomaly_detector=DEBUG --local_detectors=DEBUG
--openflow.spanning_tree=DEBUG
```

You might try to check the effectiveness of the algorithm when some switches behave as a type of "typical" L2 switch and some other can be in addition equipped with the DIAMoND algorithm:

```
sudo ./pox.py openflow.of_01 --port=6633 anomaly_detector --ttl=1 --
algo="med" l2_learning-modif log.level –WARNING
--anomaly_detector=DEBUG --local_detectors=DEBUG

sudo ./pox.py openflow.of_01 --port=6644 forwarding.l2_learning log.level
--WARNING
```

Finally, the structure of our network layer protocol is discussed and can be found in /home/mininet/pox/pox/lib/packet/bee.py, whereas further examples can be found in /home/mininet/maciej/tests/

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

DIAMoND	distributed intrusion/anomaly monitoring for nonparametric detection
MES	DIAMoND message exchange service
LAD	local anomaly detector
TIE	total impact estimation
TLS	thread level estimator
BTG	background traffic generation services
ETG	event traffic generation services
RTT	replay of traffic traces
HPA	hit peak attack
DRS-ADAM	distributed rate sharing based amplified DNS-DDoS attack mitigation
ADD	amplified DNS DDoS
RRL	response rate limit
nsIn	number of name servers
K-L	Kullback-Leibler (divergence)
TTL	time-to-live (i.e. hop distance on network topology)
DNS	domain name system
TCP	transmission control protocol
UDP	user datagram protocol

GLOSSARY OF TERMINOLOGY

analysis levels:

- local concern consensus: a subgroup where node x and at least a majority of the neighborhood of x agree on the level of concern
- local threat consensus: a subgroup where node x and at least a majority of the neighborhood of x agree on the threat level
- regional (concern/threat) consensus: defined in the same ways as local consensus, but involving a subgroup larger than the neighborhood of a single node while still smaller (at least half?) than the total population
- global (concern/threat) consensus: defined in the same ways as regional consensus, but most of the total population (85%?)

backbone network: a part of computer network that is composed of interconnected nodes

computer network: which nodes (hosts) can send packets directly to which other nodes (hosts) (i.e. the full node/edge set)

concern level: individual nodes can be "concerned" that a threat might be occurring without any data-driven analysis to support this. There can be multiple levels of concern. Concern can influence a decision about believed threat level (result of processing a local information = direct observation + concern level of the others).

host: any computer connected to a computer network

information network: the same node set as the computer network (backbone network), but the edge set represents only pairs of nodes that can communicate directly for the purpose of anomaly detection (i.e. the union of all neighborhoods for all x); this can be a (dynamically) weighted network

knowledge levels:

- direct observation from x = analysis of traffic that comes across x itself
- local information/knowledge for x = information about observed data and/or information about concern from neighbors of x
- global information/knowledge = information that could only be observed if monitoring the whole population

n-th neighbor of x: a node y is an n-th neighbor of x if the length of the shortest communication path between y and x involves n-1 intermediate nodes (note, y is a neighbor of x is implicitly equivalent to y is a 1-neighbor of x)

neighborhood of x: for a node x, the set of nodes (not including x) that x includes in assessing the threat level

node: switch/router

population: the entire set of all nodes

sensitivity: how good we are at noticing actual problems: rate of detection of true positives = TP/(TP+FN)

specificity: how confident we are that detection of a problem is because it's true: = 1-FP/(FP+TN)

stability: applies to level of concern, level of threat, and consensus; the number of time-steps over which the factor remains (mostly?) unchanged

threat level: the data-driven assessed level of likelihood that an anomaly is occurring; this will be analyzed at each node, possibly by subgroups, and also possibly at the population level (comes from direct observation)