Anomaly detection through information sharing under different topologies

Gallos, Lazaros K.; Korczynski, M.T.; Fefferman, Nina H.

DOI
10.1186/s13635-017-0056-5

Publication date
2017

Document Version
Final published version

Published in
Eurasip Journal on Information Security

Gallos et al. EURASIP Journal on Information Security (2017) 2017:5
DOI 10.1186/s13635-017-0056-5

EURASIP Journal on Information Security

RESEARCH Open Access

Anomaly detection through information sharing under different topologies

Lazaros K. Gallos[1,2], Maciej Korczyński[3] and Nina H. Fefferman[2,4,5*]

Abstract

Early detection of traffic anomalies in networks increases the probability of effective intervention/mitigation actions, thereby improving the stability of system function. Centralized methods of anomaly detection are subject to inherent constraints: (1) they create a communication burden on the system, (2) they impose a delay in detection while information is being gathered, and (3) they require some trust and/or sharing of traffic information patterns. On the other hand, truly parallel, distributed methods are fast and private but can observe only local information. These methods can easily fail to see the "big picture" as they focus on only one thread in a tapestry.
A recently proposed algorithm, Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection (DIAMoND), addressed these problems by using parallel surveillance that included dynamic detection thresholds. These thresholds were functions of nonparametric information shared among network neighbors. Here, we explore the influence of network topology and patterns in normal traffic flow on the performance of the DIAMoND algorithm. We contrast performance to a truly parallel, independent surveillance system. We show that incorporation of nonparametric data improves anomaly detection capabilities in most cases, without incurring the practical problems of fully parallel network surveillance.

Keywords: Anomaly detection, DDoS attack, Information sharing, Simulation

1 Introduction

The ability to detect anomalous behaviors, especially malicious attacks, in a network is critical for the safety of most technological systems. There are many different methods that can enhance cyber security. Typical protection strategies include the use of, e.g., firewalls or network intrusion detection systems (NIDS) [1]. An alternative approach is to attempt detecting attacks through information sharing among the nodes of such a network [2–6]. These algorithms take advantage of the fact that most cyber-attacks engage the majority of nodes in a network, so that accumulating information from other nodes can enhance the accuracy of attack detection at the local level.

For example, a widespread attack strategy in computer networks is the distributed denial of service (DDoS) attack. A large number of nodes become victims of the attacker and start emitting packets towards a given target node or a set of nodes, thus generating a lot of traffic, which necessarily influences the majority of the network [7]. If a single node can analyze traffic information from a wider area of the network, then it will have a greater probability of understanding whether a distributed attack is taking place or not. Of course, this kind of expansive surveillance can be difficult due to not only the time it takes to monitor and analyze data from large segments of the network but also in terms of negotiating access to data on traffic across other domains in the first place [8].

Based on this principle, we recently introduced [9] a local method where a node decides if it is under attack by combining its own local observations with information on its neighbors' "level of concern," a non-parametric indicator that can be shared quickly and efficiently, with little chance of violating privacy concerns. We demonstrated how this method, called DIAMoND (Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection), can lead to improvement of a node's detection capabilities. We tested the method by implementing it in a software

* Correspondence: nina.h.fefferman@gmail.com
[2]Department of Ecology, Evolution, and Natural Resources, Rutgers University, New Brunswick, NJ, USA
[4]Department of Ecology and Evolutionary Biology, University of Tennessee, Knoxville, TN, USA
Full list of author information is available at the end of the article

environment. The improvement of DIAMoND over isolated detection without information sharing was of the order of up to 20% in terms of sensitivity. However, this initial study focused only on a limited set of simplified network topologies. Since the flow of traffic is greatly influenced by topological structure of the networks itself, a critical next step is the investigation of how this method will perform under more realistic topological structures.

Here, we present a systematic simulation study of how DIAMoND behaves under controlled conditions, and we determine the influence of the algorithm parameters to improving attack detection. We stay close to the spirit of the algorithm as presented in [9, 10], but we simplify the most technical parts which are related, for example, with the nature of the packets transferred. To explore whether there are particular types of network topologies that enhance or inhibit DIAMoND's anomaly detection capabilities, we generate different model topologies that modify the connection patterns between nodes. We also study the DIAMoND algorithm on a real Internet topology, using a known configuration of AS-level connected routes [11]. We assume that a given percentage of the nodes are compromised and take part in the attack. Each node then uses the local detection algorithm, based on the traffic it observes and the concern level of its neighbors, in order to determine if it is under attack or not. We also explore the performance of DIAMoND across network topologies under different conditions of "normal" network traffic to determine how the interactions between underlying topology and expected variation in packet distribution and flow might alter detection capabilities.

2 Methods

2.1 Network 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 [12], where nodes are connected randomly to $\langle k \rangle = 3$ other nodes; (c) a scale-free network [13] created with the configuration model [14], 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 [11]; and (f) the CAIDA Autonomous System graph for May 2007 [11].

2.2 Internet traffic

Using each one of the above network topologies, we simulate traffic processes through the network. We assume that the network represents the structure of, e.g., a computer network, so that traffic is made up by packets transferred via the network links. The total count of packets on a node is determined by the sum of the "regular" traffic counts and the DDoS packets that pass through the node.

In the absence of DDoS attack, we assume that the "regular" traffic $g_i(t)$ 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.

2.2.1 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 $g_i(t)$ on each node i is selected from a Gaussian distribution (μ_i, σ_i).

2.2.2 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 $g_i(t)$ on a node is uniformly selected in the range $[\mu_i - \sigma_I, \mu_i + \sigma_i]$.

2.2.3 Exponential

In this case, a random number is drawn from an exponential distribution $\lambda \exp(-\lambda r)$ with parameter $\lambda = 1/\beta$, where β is chosen uniformly in the range β:[750:1250].

2.3 DDoS attack

During a DDoS attack, we select a percentage of the network nodes. We vary this percentage in the range from 5 to 60% of the total network. These nodes are considered compromised, and they take part in a coordinated attack. A random node is chosen in the network, and it serves as the target of the attack. All the compromised nodes emit a packet at each time step directed towards this target node. The size of this packet can vary

between $z = 1$ and $z = 200$ (in the same arbitrary units as used in the normal traffic distributions).

The routing of packets in the Internet is a complicated process that involves many protocols and a number of decisions based on the current state of the network. Here, we necessarily simplify this process. At each time step, a packet moves to a neighbor node via a routing protocol, which is based on a biased random walk [15]: with probability 70%, the packet follows the shortest path from its current location towards the target node and with probability 30%, it uniformly selects a random neighbor of its current location. This protocol guarantees that the target node will be reached within a finite and small number of steps, while at the same time a random path is selected in lieu of always using the shortest path. Every node that receives one or more of these packets at a given time step adds their total value to its regular traffic, i.e., the total traffic $w_i(t)$ in the node is now $w_i(t) = g_i(t) + n_t z$, where n_t is the number of packets on the node at time t. When the target node receives the DDoS packets, its traffic increases accordingly for that step, but at the next step these packets are discarded rather than forwarded to any other nodes.

Whenever, at a given time, one or more of the DDoS packets are located on a given node, this node is considered to be under attack, i.e., it is classified as a positive hit. Nodes with no DDoS packets are classified as negative.

2.4 The detection algorithm

The key idea behind the algorithm is that a node tries to infer the existence of attacks by combining information on its own traffic and on the perceived state of its neighbors, without knowing if its neighbors are under attack or not and without being able to analyze any direct information about the patterns in traffic its neighbors experience.

Each node uses two sensitivity thresholds to characterize its current traffic. The lower sensitivity threshold s_{iL} corresponds to the point where normal traffic w_i is $P(w_i > s_{iL}) = 6.681\%$ of the time and the upper sensitivity threshold s_{it}, which initially is defined by $P(w_i > s_{i0}) = 0.135\%$. Of course, the values for these parameters can be varied to tune the method to different sensitivity. For the Gaussian distribution, these points correspond to $s_{iL} = \mu_i + 1.5\sigma_i$ and $s_{i0} = \mu_i + 3\sigma_i$. Every node perceives its current state as either "normal", "concerned", or "attacked", according to how its current traffic count compares with these two thresholds (Fig. 1). When traffic is higher than s_{it}, then the node state is "attacked". When traffic is lower than s_{it} but still exceeds the lower threshold, i.e., $s_{it} > w_i(t) > s_{iL}$, then the state of the node switches to concerned and it becomes normal when $w_i(t) < s_{iL}$. The upper threshold, which is used to determine whether the node considers itself under attack, is dynamic. Whenever the traffic exceeds the upper threshold, then this threshold automatically lowers to 98% of its current value but always remains higher than the lower threshold s_{iL}. This is a necessary safety measure, because if the lower threshold remains unbound, then an attacker could decrease the adaptive threshold indefinitely and artificially induce very low specificity (almost all traffic would be classified as attack). When the node detects that it is no longer under attack, it increases the value of s_{it} to 102% of its current value, but this always remains lower or equal to its initial value s_{i0}.

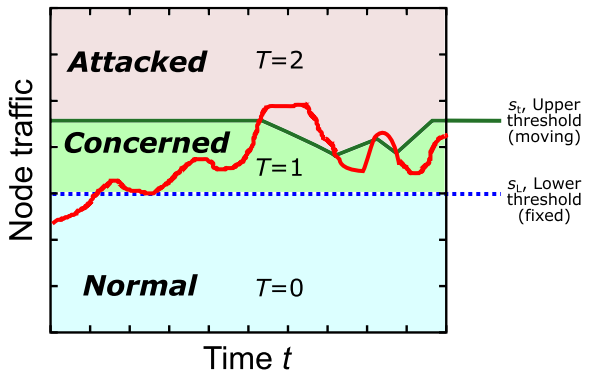

Fig. 1 Demonstration of dynamic thresholds and the perceived attack state by a node. The *red line* represents the traffic count through the node, as a function of time. The lower threshold s_L is fixed, but the upper threshold s_t decreases when traffic exceeds its value, and it increases when traffic is lower than that. In each of the three areas defined by the two thresholds, the perceived threat level of the node assumes the value of either $T = 0$, $T = 1$, or $T = 2$

A node monitors its state according to two main indicators, its threat level and its concern level. Additionally, a node has a "feeling" for the concern of its neighbors, i.e., the nodes directly physically or logically connected in the network structure, without explicitly knowing their threat level. As a result, every node keeps a set of three parameters, which helps it evaluate the state of the network and its own state. The three parameters are (a) the concern level of the node $c_i(t)$, (b) the average concern level of its neighbors expressed through the function $L_i(t)$, and (c) the observed threat level $T_i(t)$, based on internal analysis of its own traffic. All three parameters take values in {0, 1, 2} and are updated at every time step.

The threat level of a node is determined by the local anomaly detection algorithm. As mentioned above, here, we use the simplest case where the level of traffic $w_i(t)$ is directly compared to the two sensitivity thresholds and exclusively depends on their value. If $w_i(t) < s_{iL}$ then $T_i(t) = 0$; if $s_{iL} < w_i(t) < s_{it}$, then $T_i(t) = 1$; and if $w_i(t) > s_{it}$ then $T_i(t) = 2$.

The function $L_i(t)$ provides a characterization of the average concern level $\langle c_i \rangle = \Sigma_j\, c_j(t-1)/k_i$, computed over all the k_i neighbors j of node i at time $t-1$. A low concern level in the range $\langle c_i \rangle < 0.4$ leads to $L_i(t) = 0$; we set $L_i(t) = 1$ when $0.4 < \langle c_i \rangle < 1.3$; and $L_i(t) = 2$ when $1.3 < \langle c_i \rangle$.

The concern level of the node is updated according to the values of $T_i(t-1)$ and $L_i(t-1)$. At any given step, the concern level $c_i(t)$ is equal to the threat level $T_i(t-1)$ at the previous time step, except when $T_i(t-1) < 2$ and the neighborhood concern is $L_i(t-1) = 2$, in which case it becomes $c_i(t) = T_i(t-1) + 1$.

2.5 Independent detection

In order to quantify possible benefits of employing the DIAMoND algorithm, we compare the results to a similar algorithm, with the difference that we now remove all the information exchange among the nodes. In this scheme, a node detects if it is under attack by simply comparing its current traffic with a given value of s_{i0}, which remains fixed for the entire realization. Simply put, this represents a typical rate limiting algorithm, such as in Refs. [16–18]. If $w_i(t) > s_{i0}$, then the node decides that it is "attacked"; otherwise, it is in a normal state. The key difference is that the value of the upper threshold in DIAMoND changes according to the current attack state of the node and information from its neighbors, while in independent detection this threshold remains unchanged. In this way, we can evaluate the importance of a moving threshold and whether the involved calculation presents visible improvement over the simpler implementation without node communication.

The main parameter involved in the parallel algorithm is the value of the threshold, used to determine the attack state. Obviously, a very low threshold would lead a node to decide that it is constantly under attack, while a very high threshold would not be sensitive to the attacks, in other words, trading sensitivity for specificity. In order to compare the results of the parallel algorithm with DIAMoND, we studied two separate cases, where this threshold assumes either the low threshold value of DIAMoND, s_{iL}, or the upper value, s_{i0}.

2.6 Simulation details

The network size for the model systems is 10,000 nodes. The May 2004 CAIDA network has 17,160 nodes, and the May 2007 CAIDA network has 24,942 nodes. The network operates for 100 steps under normal conditions, without any attack in the system, when the traffic in all nodes is always $w_i(t) = g_i(t)$. At time $t = 100$, a percentage of nodes are compromised and emit one packet each at every time step until $t = 300$.

The predefined attack period reflects typical duration of amplification DDoS attack. As the majority of them stem from booter services [19, 20], they are usually shorter than 300 s [21]. After that, the network returns to regular traffic, and the attacker nodes stop adding new packets in the system. The DDoS packets that remain in the system continue to be transferred until they reach the target node, even if this takes place for $t > 300$. We follow the network operation for another 100 steps, i.e., until $t = 400$.

As an example, we present in Fig. 2 the traffic in the target node during such an attack. The target node obviously receives all the DDoS packets, and its traffic increases significantly during the attack period and for a few steps after it stops. In contrast, a random node in the network may not feel the attack, if the attack packets that pass through this node are not enough to raise its traffic to alarm levels. Interestingly, it takes a few steps after the beginning of the attack for the traffic to reach its maximum value, and the increased traffic effect remains for a few steps after the attack has ended.

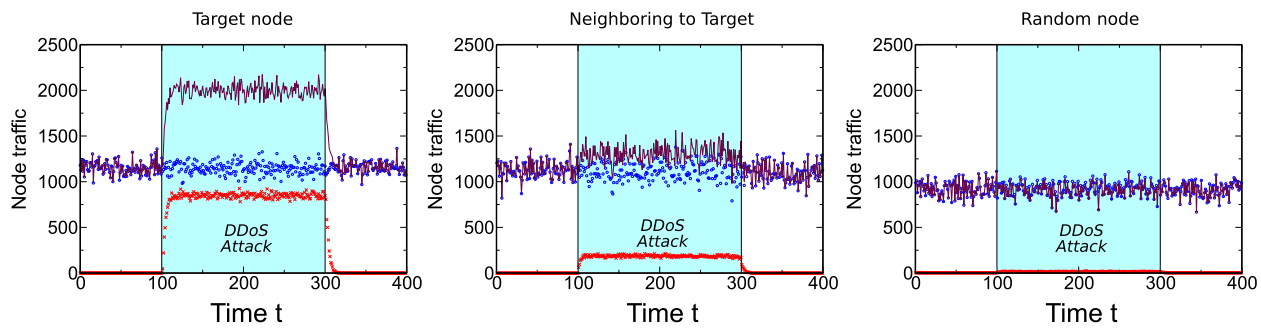

Fig. 2 Time evolution of traffic on the target node (*left*), on a node neighboring the target node (*middle*), and on a random network node (*right*). The DDoS attack took place in the range $100 < t < 300$ on the CAIDA network structure of May 2004. The DDoS attack is performed by 5% of the network nodes, and the DDoS packet size is $z = 1$. The *blue symbols* indicate the normal traffic through the node, the *red symbols* represent node traffic due to the malicious packets, and the continuous line is the total traffic seen by the node. It is clear that the target node sees an increased traffic during the attack, while traffic on the random node is not influenced significantly. There are very few nodes with intermediate traffic, and they are all in close proximity to the target node

2.7 Evaluation of the results

We use the standard measures of accuracy, sensitivity, specificity, and precision, which are defined as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$
$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

where TP = true positive, TN = true negative, FP = false positive, and FN = false negative.

In previous work [16], the intensity of an attack was determined in practical terms through the number of unsuccessful TCP connections per source or per destination. In this study, we instead present our results as a function of the two main parameters that determine the attack "intensity", (a) the fraction of compromised nodes p and (b) the packet size z. The value of p indicates what percentage of the network participates in the attack, while the packet size z shows how "intense" is the attack from each node. For example, if the product pz is constant, then it is possible that we have many nodes that emit a small number of packets or that we have a few nodes that emit a large number of packets.

We calculate each of the four main indicators, e.g., accuracy, for the parallel (1st column) and the DIAMoND (2nd column) algorithms, averaged over the attack stage ($120 < t < 300$). In the plots below, we assign a color that corresponds to the indicator and ranges from blue (50%) to red (100%). When a value is lower than 50%, the algorithm fails completely, and we indicate this with black.

The improvement or loss in each indicator when we use DIAMoND versus the parallel algorithm is shown in the 3rd column of each plot. In these plots, we show the relative increase as:

$$\text{Gain} = \frac{A(\text{DIAMoND}) - A(\text{Parallel})}{A(\text{Parallel})},$$

where A can represent accuracy, sensitivity, specificity, or precision. Yellow colors in these plots indicate that DIAMoND outperforms the parallel version (and red that it significantly outperforms it), while the parallel version performs better when the color is blue. The areas where both algorithms yield comparable results, i.e., within 2% of each other, and there is no clear improvement of using one over the other, are shown in gray.

3 Results and discussion

3.1 Gaussian-distributed traffic

In our first simulation, we use a Gaussian distribution for the normal traffic on each node. The results in Fig. 3 indicate that both DIAMoND and the parallel algorithm can in general detect the attack successfully, independently of the underlying topology. 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. It is worth noting that 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.

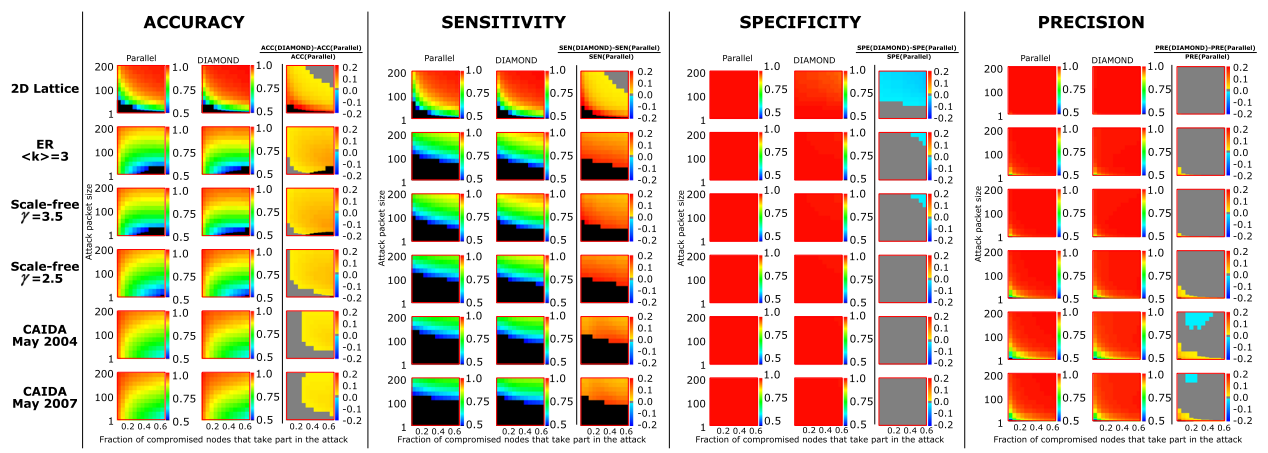

Fig. 3 Values of accuracy, sensitivity, specificity, and precision for Gaussian distributed traffic. For each index, the first column shows the results for the parallel algorithm, the second column shows the results for DIAMoND, and the third column displays the relative gain of the DIAMoND algorithm over the parallel algorithm. We use the upper threshold, s_{10}, as the threshold for the parallel algorithm

In terms of accuracy, we here find that the DIAMoND algorithm significantly outperforms the 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, s_{i0}. We examined how this comparison changes when the parallel threshold takes the value of the lower threshold, s_{iL}. As shown in Fig. 4, 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.

Our descriptive analysis of the numerical results shows that the average values of our measures converge satisfactorily within a margin of a few percent. Since the large number of simulation conditions makes it very difficult to present the confidence intervals for all of our results, we instead present a sample of 95% confidence intervals for the accuracy, where we use a combination of the p and z parameters: $p = 0.05$ and $p = 0.2$ with $z = 10$ and $z = 120$, for all six network topologies (Fig. 5). The confidence intervals are quite narrow, relative to the means. These results are representative of the results from the entirety of our simulations. (Note that traditional frequentist statistical tests, e.g., p values, are inappropriate since significance of any level can be achieved simply by increasing the number of simulation replicates performed.)

3.2 Uniformly distributed traffic

In order to determine the efficiency of the DIAMoND algorithm under varying conditions, we studied additional regular traffic patterns. Here, we assume that network

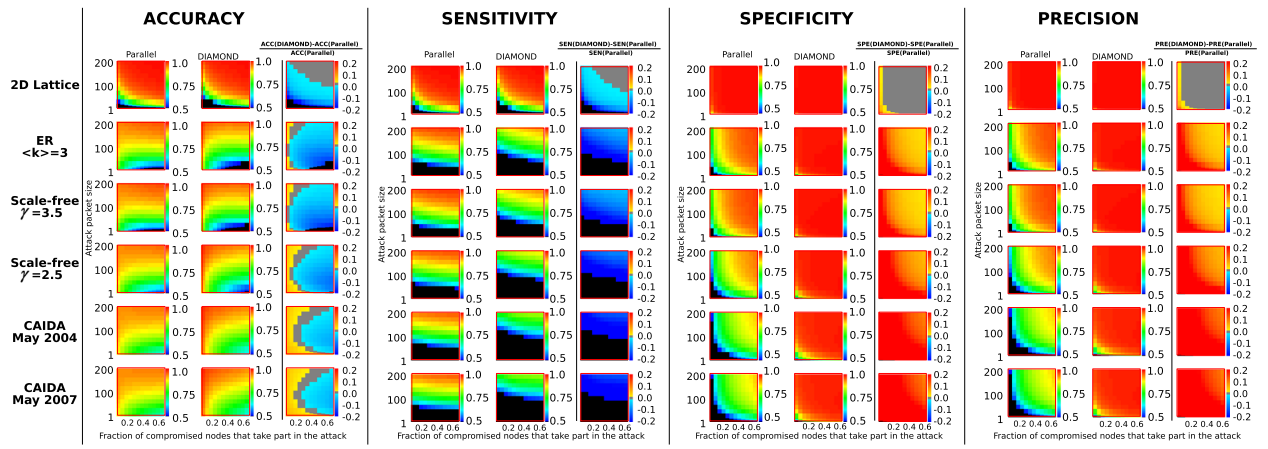

Fig. 4 Results for Gaussian traffic. Same as Fig. 3, but now for the parallel algorithm, we use the lower threshold, s_{iL}

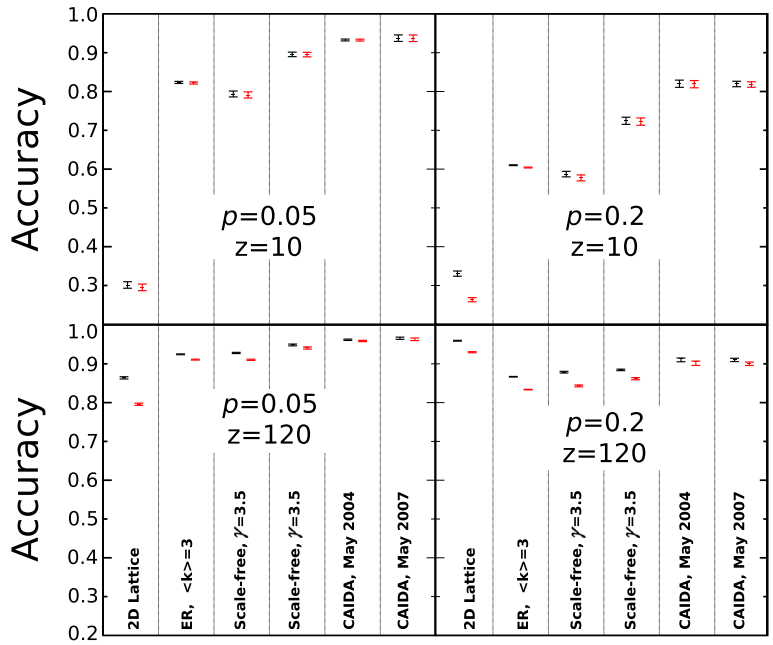

Fig. 5 The 95% confidence intervals of accuracy for Gaussian traffic. The results refer to the simulation conditions of Fig. 3. The *p* and *z* values are shown on the plots. The results for DIAMoND are shown in *black*, and the results for the parallel algorithm are shown in *red*

traffic in a node follows a uniform distribution with time, as described in the Section 2. The threshold values are different than in the case of Gaussian distribution, because they have been selected to correspond to the same point in the cumulative distribution function. This means that, in the absence of attack, the same percentage of regular traffic is mislabeled as attack packets, independently of the traffic distribution.

In general, the accuracy improves significantly, both in DIAMoND and the no-sharing algorithms (Fig. 6). 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*.

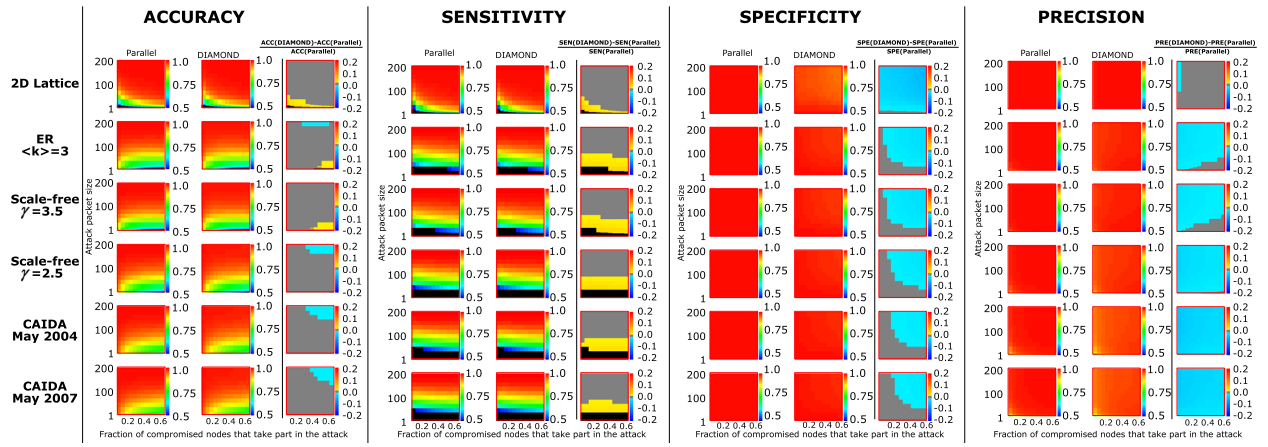

Fig. 6 Values of accuracy, sensitivity, specificity, and precision for uniformly distributed traffic. For the parallel algorithm, we use the upper threshold, s_{i0}

The increased detection efficiency, compared to Gaussian traffic in Fig. 3, 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.

In the above discussion, the threshold for the parallel algorithm was fixed to the upper DIAMoND threshold value, s_{iO}, and there was no clear advantage of one method over the other. When we move the parallel threshold to the lower value, s_{iL}, then DIAMoND outperforms the parallel algorithm again (Fig. 7). 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).

3.3 Exponentially distributed traffic

As an extreme example of inhomogeneous traffic, we studied the case where traffic through a node follows an exponential distribution over time. 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. 8. 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.

The differences between the two algorithms are not significant, even though DIAMoND outperforms the parallel algorithm in precision. When we lower the threshold (Fig. 9), 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.

4 Conclusions

We performed an expanded analysis of a recently introduced algorithm (DIAMoND) for attack detection through nonparametric information sharing. We simulated different conditions, with the aim of understanding the performance of the algorithm under a controlled environment. As a result, the algorithm 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 algorithm performed exceptionally well under almost all the metrics and independently of the simulation conditions, when traffic was gaussian or uniformly distributed. In the extreme case of exponential distribution, DIAMoND did not manage to maintain its accuracy, mainly because it could not detect the majority of positive hits, even though identified positive hits were almost never false.

The results of the algorithm were evaluated against a parallel algorithm, where there was no information

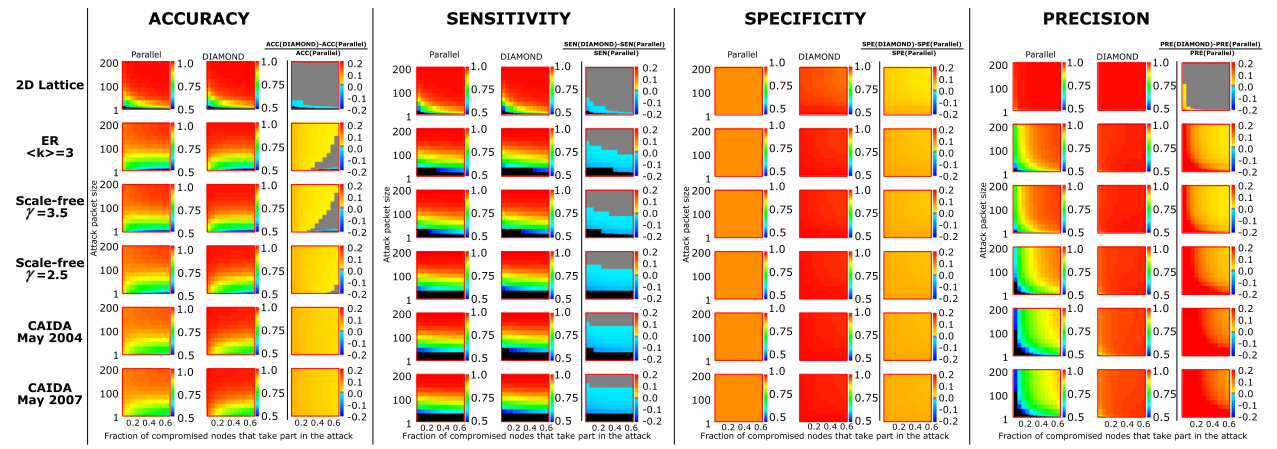

Fig. 7 Results for uniform traffic. Same as Fig. 6, but now for the parallel algorithm, we use the lower threshold, s_{iL}

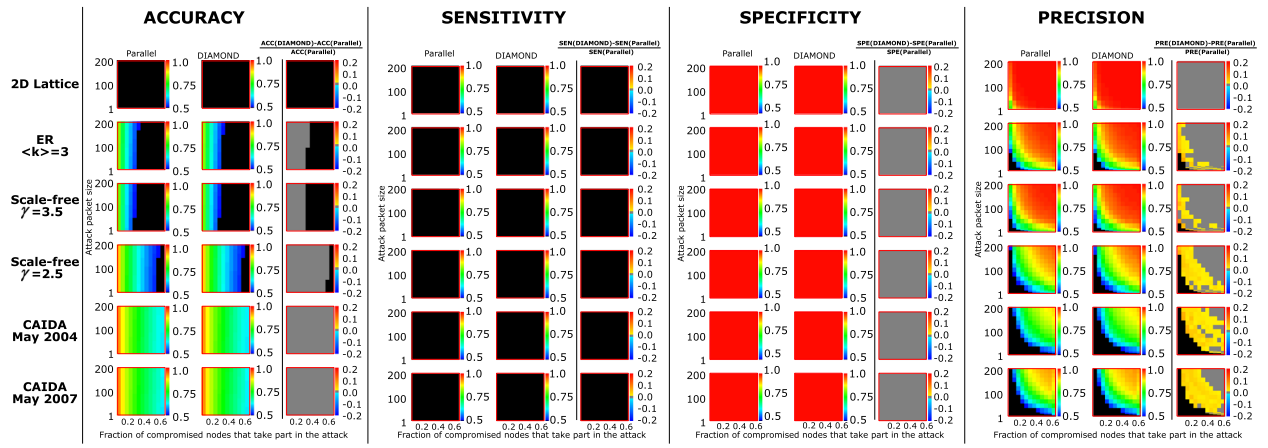

Fig. 8 Values of accuracy, sensitivity, specificity, and precision for exponentially distributed traffic. For the parallel algorithm, we use the upper threshold, s_{iO}

sharing among the participating nodes. In this case, traffic in a node is compared to a pre-defined threshold, in order to decide if the node is under attack or not. In the majority of the cases, DIAMoND outperformed the parallel algorithm by a wide margin. In a few cases, the opposite was actualy 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.

In summary, we found that sharing a general 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.

The results presented here explored only the impact of network topology; however, earlier work considered the performance of DIAMoND on simplified topologies but under more realistic scenarios of network traffic [9, 10]. These same studies also explored the impact of having only partial participation among network nodes (where the rest of the nodes employed purely parallel detection and did not use or share non-parametric information). Results demonstrated improvement over purely parallel systems, and that the majority of performance gain can be achieved in simplified networks with only 30% node participation. These features have now been explored independently, but the emergent behavior of any self-organizing system depends on the synergistic behavior of all the elements acting together. Now that we have a foundational understanding of the influence of these features on

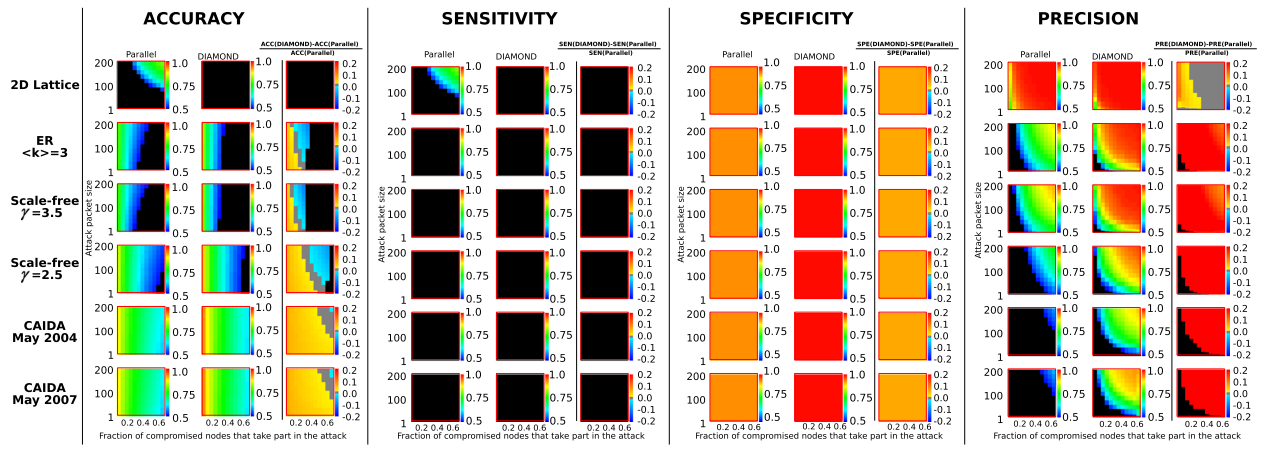

Fig. 9 Results for exponential traffic. Same as Fig. 8, but now for the parallel algorithm, we use the lower threshold, s_{iL}

DIAMoND system performance, the immediate next step for future work will be to explore the combined impact of these features.

Acknowledgements
This material is based upon work supported by the NSF under Grants CNS-1646856 (LKG) and CNS-1646890 (NHF). This research was also supported by the US Department of Homeland Security sponsored under the Air Force Research Laboratory (AFRL) agreement number FA8750-12-2-0232.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]DIMACS, Rutgers University, Piscataway, NJ, USA. [2]Department of Ecology, Evolution, and Natural Resources, Rutgers University, New Brunswick, NJ, USA. [3]Faculty of Technology, Policy and Management, Delft University of Technology, Delft, The Netherlands. [4]Department of Ecology and Evolutionary Biology, University of Tennessee, Knoxville, TN, USA. [5]Department of Mathematics, University of Tennessee, Knoxville, TN, USA.

Received: 16 November 2016 Accepted: 16 February 2017
Published online: 28 February 2017

References

1. Meng, W, & Li, W (2015). In Networking for Big Data, ed. by S Yu, X Lin, J Misic, X Shen, (p. 195), CRC Press, Taylor & Francis'.
2. V Yegneswaran, P Barford, S Jha, in Proceedings of NDSS, 2004
3. P Chhabra, C Scott, ED Kolaczyk, M Crovella, in Proceedings of INFOCOM, IEEE, 2008
4. MITRE, Structured Threat Information eXpression: a structured language for cyber threat intelligence information. [Online]. Available: https://stix.mitre.org. Accessed 27 Feb 2017.
5. MITRE, Trusted Automated eXchange of Indicator Information: enabling cyber threat information exchange. [Online]. Available: https://taxii.mitre.org. Accessed 27 Feb 2017.
6. CybOX, Cyber Observable eXpression [Online]. Available: https://cybox.mitre.org. Accessed 27 Feb 2017.
7. J Mirkovic, P Reiher, SIGCOMM Comput Commun Rev **34**, 39 (2004)
8. Serrano, O, Dandurand, L, Brown, S (2014). In Proceedings of the ACM Workshop on Information Sharing & Collaborative Security. ACM.
9. M Korczyński, A Hamieh, JH Huh, H Holm, SR Rajagopalan, NH Fefferman, in Proc. of IEEE ICCCN, 2015
10. M Korczyński, A Hamieh, JH Huh, H Holm, SR Rajagopalan, NH Fefferman, IEEE Communications Magazine **54**, 60 (2016)
11. J Leskovec, J Kleinberg, C Faloutsos, in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2005
12. Erdos, P, & Renyi, A (1960). Magyar Tud. Akad. Mat. Kutato Int. Kozl. 5, 17.
13. AL Barabasi, R Albert, Science **286**, 509 (1999)
14. Molloy, M, & Reed, BA (1995). Random Structures and Algorithms 6, 161.
15. L Skarpalezos, A Kittas, P Argyrakis, R Cohen, S Havlin, Physical Review E **88**, 012817 (2013)
16. M Korczyński, L Janowski, A Duda, in Proc. of IEEE ICC, 2011
17. H Wang, D Zhang, KG Shin (2002). In Proc. of IEEE INFOCOM, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE.
18. Deal, R (2004). Cisco Router Firewall Security. Cisco Press.
19. L Kramer, J. Krupp, D Makita, T Nishizoe, T Koide, K Yoshioka, C Rossow, in Proc. of RAID.2015
20. JJ Santanna, R Durban, A Sperotto, A Pras, in Proc. of IFIP/IEEE IM, 2015
21. A Noroozian, M Korczyński, CH Ganan, D Makita, K Yoshioka, M van Eeten, in Proc. of RAID. 2015