Making Active-Probing-Based Network Intrusion Detection in Wireless Multihop Networks Practical: A Bayesian Inference Approach to Probe Selection

Rodrigo do Carmo, Justus Hoffmann, Volker Willert, and Matthias Hollick

***Abstract*—Practical intrusion detection in Wireless Multihop Networks (WMNs) is a hard challenge. The distributed nature of the network makes centralized intrusion detection difficult, while resource constraints of the nodes and the characteristics of the wireless medium often render decentralized, node-based approaches impractical. We demonstrate that an *active-probing-based network intrusion detection system* (AP-NIDS) is practical for WMNs. The key contribution of this paper is to optimize the active probing process: we introduce a general Bayesian model and design a probe selection algorithm that reduces the number of probes while maximizing the insights gathered by the AP-NIDS. We validate our model by means of testbed experimentation. We integrate it to our open source AP-NIDS *DogoIDS* and run it in an indoor wireless mesh testbed utilizing the IEEE 802.11s protocol. For the example of a selective packet dropping attack, we develop the detection states for our Bayes model, and show its feasibility. We demonstrate that our approach does not need to execute the complete set of probes, yet we obtain good detection rates.**

***Index Terms*—Bayes inference, Security, Intrusion Detection, Wireless Multihop Networks**

I. Introduction

COMMUNICATIONS in Wireless Multihop Networks (WMNs) run on top of a decentralized, wireless, and cooperative infrastructure. Because of their decentralization, WMNs are harder to monitor and to control than traditional centralized wired networks. However, a reliable multihop infrastructure is desirable for most applications. Several attacks can drastically harm a multihop network; for instance, a properly located attacker might carry out an easy-to-implement selective packet dropping, and this is both nocuous and difficult to detect [1]. For this reason, the security of the underlying infrastructure is specially important in these kind of networks.

The nodes in a WMNs are, generally, not accessible for centralized management, and they are very often resource constraint. Therefore, local Intrusion Detection Systems (IDS) are unpractical for these networks. In addition, passive eavesdropping limits the range of attacks that can be detected [2]. In previous work, a practical conceptually different alternative has been presented for intrusion detection: the so-called *active probing* [3], [4]. By transmitting testing packets to the nodes and analyzing their response, an Active-Probing-Based Network IDS (AP-NIDS) achieves intrusion detection while conserving the resources of the nodes. It is deployed on a single, trusted node which serves for intrusion detection.

Active probing for intrusion detection is a promising approach in WMNs, yet making it completely functional remains an open challenge. An AP-NIDS is provided with a set of probes, and each probe aims at unveiling the presence of a particular attack. Transmitting the complete set of probes is, clearly, not wise because it increases the network overhead, detection time, and most of the attacks are unlikely to happen simultaneously. For this reason, creating a mechanism that uses a logic in order not to transmit the complete set of probes is necessary.

Rish et al. proposed a probe selection mechanism for fault determination in distributed systems [5], [6]. Their idea is conceptually similar to what we need for AP-NIDS but it is intended for determining faults in complete systems, as a result of probing their individual components. Each component has only two states, namely, "up" or "down", and the system state is inferred from the state of all of its components. In an AP-NIDS, however, the scenario is different. The performed probings are reduced to single individual nodes, their state being not only "up" or "down" but all the different possible attacks. Thus, in this work we reformulate the approach presented in [5], [6] to make it applicable to active probing intrusion detection. Our approach is, nevertheless, valid for other applications where individual nodes have to be probed against different states.

The contributions of this paper are as follows.

- We model a temporal-selective Bayes classifier to infer the state of a network node under test. It classifies whether a node misbehaves based on the outcome of a set of active probes. To implement this classifier, we design a recursive probe selection scheme. It is based on the current posterior of the Bayes classifier and a prediction step, and facilitates to reduce the number of active probes while maximizing the insights gained by the set of probes executed.
- We integrate the proposed mechanisms into our open source AP-NIDS *DogoIDS*[1]. We perform extensive experimentation in an indoor mesh testbed using the IEEE 802.11s protocol for evaluating the performance of our proposed Bayesian classifier. We perform our evaluation

Rodrigo do Carmo, Justus Hoffmann, and Matthias Hollick are with the Secure Mobile Networking Lab (SEEMOO) at TU Darmstadt, Mornewegstr. 32, 64293, Darmstadt, Germany. Email: {rdocarmo, jhoffmann, mhollick}@seemoo.tu-darmstadt.de.

Volker Willert is with the Control Theory and Robotics Lab at TU Darmstadt, Landgraf-Georg-Str. 4, 64283, Darmstadt, Germany. Email: vwillert@rtr.tu-darmstadt.de.

[1]DogoIDS is available at http://sourceforge.net/projects/dogoids.

978-1-4799-3780-6/14/$31.00 ©2014 IEEE

with real attacks.

The rest of this paper is organized as follows. Section II describes related work. In Section III, we introduce the necessary background on active-probing intrusion detection, and define our adversary model. Section IV presents the Bayesian inference model proposed in this paper. In Section V, we describe the experiments performed to evaluate our model, and Section VI presents and discusses the results obtained. Finally, Section VII concludes this work.

II. Related work

Rish et al. proposed a probe selection mechanism for fault determination in distributed systems [5], [6]. They employ a two-layer Bayesian network where a prior distribution over the states of a system is employed to select the most informative probes, and the belief is updated on each test. Although this is an interesting approach for fault localization, in this paper we reformulate it for intrusion detection. We use a naive Bayes classifier. The components of our network (the nodes) are individually tested and therefore not interconnected in a Bayes network. The state of a node is not only "up" or "down" but also the attack/misbehavior present. The outcome of our active probes is a vector that contains the probability of a node to be in a certain state. In addition, we adapt the algorithm for probe selection in [5]. We do not employ an information theoretic approach that links the probes with a dependency matrix, but a search algorithm that looks into the distributed probabilities of the attacks.

Regarding intrusion detection in wireless multihop networks, many schemes propose the deployment of sensors in all or a great part of the network nodes [7], [8], [9], [10], but none of them is validated in practice. There exists work that is experimentally validated, such as AODVSTAT [11], LiPaD [12], or OpenLIDS [13]. These implementations, however, are expensive for the network nodes in terms of resources. Some other works propose modifying the routing protocols [14], [15]. These are not validated in practice and therefore we argue that modifying the protocols currently employed and standardized might be a reasonable approach only if the modification notably improves the attack detection/repudiation and if they are validated. In previous work, we presented an alternative to intrusion detection which does not need to be deployed in every node [3], [4]. We deployed a proof-of-concept and showed that it is practical for WMNs.

III. Background: Active-Probing Intrusion Detection

In this paper we propose a naive Bayes classifier to infer the state of component under test based on the outcome of a sequence of measurements, or probes. Our approach is general enough and can be applied to a variety of situations. However, we are also interested in applying our model to a concrete case: attack/misbehavior inference in an active-probing-based network intrusion detection system. For this reason, in this paper we model our classifier in the context of an AP-NIDS. We show how we can theoretically model the classifier and, at the same time, how it can be applied to solve a particular problem. In order to understand the active probing mechanism, we describe in this section the basics of an AP-NIDS together with the adversary model we employ.

A. Adversary Model

Capabilities of the attacker. We assume that the attacker is capable of either introducing one or multiple malicious nodes, or of taking control of one or multiple legitimate nodes of the network (byzantine behavior). In both cases we assume the malicious nodes to be internal to the network, i.e., the nodes belongs to the network. The attacker is active and, hence, can freely communicate with the legitimate nodes. In particular, it can create, manipulate, send, receive, forward, and drop packets.

Limitations of the attacker. We assume that the attacker cannot launch attacks on the physical layer (like jamming attacks), and it can only communicate with the nodes within its transmission range. The attacker cannot increase the transmission power or improve its sensitivity. The computation and power capacity of the attacker are limited to the resources of the nodes that it is controlling. In addition, the attacker cannot change the physical location of the compromised nodes. The attacker's purpose is to degrade the quality of service of the network by selectively dropping packets [3].

B. The Active Probing Mechanism

In a wireless multihop network there is no clear line of defense; the network is created ad hoc and the nodes collaborate to forward the packets to their destination without the need of a central management unit. Hence, the intrusion detection should be distributed in the network. Since we do not want to harm the resources of the nodes by deploying IDS sensors on them, we utilize an active-probing-based IDS which is deployed in single node (or more) dedicated for intrusion detection which can also be mobile. We consider that the mobility of the AP-NIDS node is out of the scope of this paper.

The active probing technique uncover malicious nodes by creating and transmitting testing packets. The testing packets are transmitted within the context of a *probe* that is defined as follows: *a probe is the set of steps and testing packets involved to detect one particular attack*. For example, a probe can be created to detect "selective dropping of HTTPS packets". To do so, it might send several different testing packets. Although a probe is well-defined to detect one particular attack, it also holds information about other attacks. A detected attack could be a subcategory of a more general attack category or vice versa, hence, it is obvious that the observations of different probes are correlated. For example, if a probe aims at detecting the dropping of HTTPS packets, some of the testing packets might establish information pointing to a more general attack dropping all TCP packets.

After the testing packets are sent to the *target node*, which is the node being tested, the AP-NIDS gathers the traffic generated by this node (if any) and analyzes it. The testing packets are assumed to be indistinguishable from regular packets in the network to conceal the IDS from attacker nodes.

The reaction of the nodes to these packets depends on the packets themselves and on the chosen protocol. For example, data packets might need to be forwarded to neighboring nodes while path discovery packets are replied according to the specification of the protocol. In order to accomplish the active-probing-based intrusion detection, the AP-NIDS performs three general steps:

1) Selection of the mesh node to be probed, 2) sending of testing packets, and 3) analysis of the data generated by the target node after the active probing.

After completing the above steps, the AP-NIDS infers that: 1) Either the target node behaved as expected and no attack is detected, 2) an attack was detected and classified, or 3) the IDS cannot assess the target node sufficiently [3].

Given that an AP-NIDS is provided with a potentially large set of probes, launching all of them sequentially until an attack is detected is not resource efficient. To reduce the overhead of active probing, in the next section we propose a mechanism based on Bayesian inference to reduce the number of probes.

IV. Bayesian Inference Model for Active-Probing Intrusion Detection

In this section, we describe the temporal-selective Bayesian classifier to optimize the active probing process. We first concisely describe the basics of the naive Bayes classifier, which we then extend to the temporal domain. We further introduce a novel feature selection strategy for our classifier.

A. Definitions

Let $\mathcal{N} = \{n_i\}_{i=0}^{I}$ be a set of $I+1$ class labels/hypotheses, which are in our case the different attacks to be detected by the AP-NIDS. One of the classes (labeled n_9 in our experiments) represents "normal condition", and stands for the detection result "no attack detected". Let $N_x \in \mathcal{N}$ be the discrete state of the target node at position x in the network that is analyzed by the AP-NIDS. Let $\Phi = \{\phi_j\}_{j=1}^{J}$ be a set of binary features $\Phi \in \{\text{true}, \text{false}\}^J$. In our case each feature ϕ_j equals the result of a probe j applied to the target node, and is designed to detect one specific attack n_i . From now on, we assume there is only one specific probe per attack and for each attack there exists a probe. The probe j corresponds to attack n_i if $i = j$. The different attacks to be investigated by the AP-NIDS are, for example, n_1 "dropping of ARP packets" or n_2 "dropping of DNS packets".

B. Bayesian Model for Attack Detection

Our goal is to infer whether the target node is attacked or not and—in case of attack—which kind of attack has taken place. One simple way to detect an attack is to apply all probes to the target node, look at all feature vectors, and choose the probe labeled with a "true". However, this method has a number of drawbacks. First, what to decide if several probes lead to a positive answer. This usually happens because it is typically not possible to design probes that lead to an unambiguous classification result. Second, we have to execute all probes, which causes a large amount of traffic, is computationally expensive, and time consuming. We propose to follow a probabilistic detection approach to reduce this overhead. Now, the AP-NIDS provides conditional probabilities $p(\phi_i|N_x)$ for a specific feature of a probe ϕ_i given a specific attack $N_x = n_i$. This tells how uncertain the AP-NIDS is for its feature given an specific attack. Here, $p(\phi_i|N_x)$ is a discrete conditional probability that can be summarized in a Conditional Probability Table (CPT) with probabilities $p(\phi_i = \text{true}|N_x = n_i) = p_{ti}$ and $p(\phi_i = \text{false}|N_x = n_i) = p_{fi}$ for all possible attacks $n_i \in \mathcal{N}$ and binary feature values $\phi_i \in \{\text{true}, \text{false}\}$ shown in Table I.

Table I
Conditional Probability Table $p(\phi_i|N_x)$

ϕ_i	n_1	n_2	n_3	n_4	n_5	n_6	...	n_i	...	n_I
true	p_{t1}	p_{t2}	p_{t3}	p_{t4}	p_{t5}	p_{t6}	...	p_{ti}	...	p_{tI}
false	p_{f1}	p_{f2}	p_{f3}	p_{f4}	p_{f5}	p_{f6}	...	p_{fi}	...	p_{fI}

Each probe results in a certain CPT that has to be either learned from data or constructed empirically. It satisfies $p(\phi_i|N_x) \geq 0$ and $\sum_{\phi_i} p(\phi_i|N_x) = 1$. Assuming that the features ϕ_i of the different probes are statistically independent, $p(\Phi|N_x) = \prod_i p(\phi_i|N_x)$, and introducing a prior probability $p(N_x)$, we are able to infer the posterior probability $p(N_x|\phi_i)$ for each probe individually via Bayes' rule:

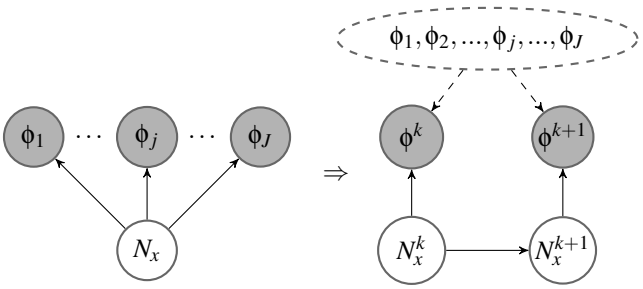
$$p(N_x|\phi_i) \propto p(\phi_i|N_x)p(N_x)\,. \tag{1}$$

Combining all individual features to one common posterior probability, we arrive at the naive Bayes classifier [17] as depicted in Fig. 1 a):

$$p(N_x|\Phi) \propto p(N_x)\prod_{j=1}^{J} p(\phi_j|N_x) \tag{2}$$

This classifier fuses all the different information sources Φ, namely the features of the different probes, to one common output. Which attack $n_\star$ has the biggest probability to be the attack really present can be found by maximizing the posterior probability as follows:

$$n_\star = \arg\max_i p(N_x = n_i|\Phi)\,. \tag{3}$$



a) Naive Bayes classifier b) Temporal-selective Bayes classifier

Figure 1. a) Graphical model of the naive Bayes classifier and b) the proposed temporal selective Bayes classifier. N_x^k is the state of the target node x and ϕ^k is the selected probe at time step k.

Using this approach, we are able to express uncertainty about the features and the classification result, and consider the statistical dependency between attacks. In addition, we can account for prior knowledge modeled with the prior $p(N_x)$ preferring specific attacks. But still all probes have to be taken into account. Therefore, we unroll the naive Bayes classifier along discrete time and only allow for one probe $\phi^k \in \Phi$ to be executed at each time step k. This leads to a first order Markov chain [16] for the state of the target node as depicted in Fig. 1 b). Hence, the state of the target node N_x^k also becomes time dependent and we arrive at a time dependent posterior:

$$p(N_x^{k+1}|\phi^{1:k+1}) \propto p(\phi^{k+1}|N_x^{k+1}) \underbrace{\sum_{N_x^k} p(N_x^{k+1}|N_x^k) p(N_x^k|\phi^{1:k})}_{p(N_x^{k+1}|\phi^{1:k})}, \tag{4}$$

where $\phi^{1:k}$ is the set of all probes $\phi^{1:k} = (\phi^1, \phi^2, ...\phi^k)$ executed up to time k. This Markov chain is completely determined by the initial prior probability $p(N_x^0)$, the transition probability $p(N_x^{k+1}|N_x^k)$ and the observation likelihood $p(\phi^k|N_x^k)$ applying one specific probe $\phi^k = \phi_i$ at each time step k. Assuming the transition probability to be the identity matrix, there is no uncertainty introduced because of the state transition and for the temporal prior/prediction it follows $p(N_x^{k+1}|\phi^{1:k}) = \sum_{N_x^k} p(N_x^{k+1}|N_x^k) p(N_x^k|\phi^{1:k})$. Hence, the Markov chain equals a stepwise executed naive Bayes classifier. Additional sources of information can easily be introduced, e.g., knowledge about correlations between attacks that have already been detected in neighboring nodes and the attack to be detected in the current node. However, up to now, we do not know which probe ϕ_i to execute at which time step k. It thus remains open: *how many probes to execute and in which temporal order should they be executed to yield a reliable detection result?*

Our selection mechanism is a two step procedure that is executed each time after a new posterior has been calculated. We define two sets of probes with their elements changing over time. The first set $\Gamma^- := \phi^{1:k}$ is the set of all the probes executed up to time step k. The second set $\Gamma^+ := \Phi - \Gamma^-$ is the set of the remaining probes that can still be executed in the future. First, the current posterior is maximized following Equation 3

$$n_\star^k = \arg\max_i p(N_x^k = n_i|\Gamma^-). \tag{5}$$

Now, the attack $n_\star^k$ the AP-NIDS assumes to be most probable is time dependent. If this maximum probability exceeds a given threshold θ at time k

$$p(N_x^k = n_\star^k|\Gamma^-) > \theta, \quad \Rightarrow \quad n_{\text{final}} = n_\star^k, \tag{6}$$

the AP-NIDS is certain enough about its classification result, the temporal selective naive Bayes classifier stops, and the final decision is n_{final}. If Equation 6 is not fulfilled, the AP-NIDS selects a new probe ϕ^{k+1} to be launched based on the following optimization criterion:

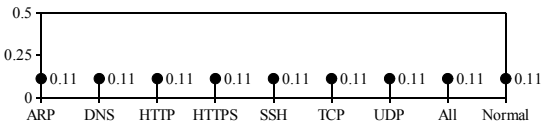
Algorithm 1: Probe selection and state update algorithm

Input: State N_x^k of the target node
Output: Updated state N_x^{k+1} of the target node
$\Phi \longleftarrow$ available set of probes;
$\Gamma^- \longleftarrow \emptyset$;
while $n_\star^k < \theta$ **and** $\Gamma^+ \neq \emptyset$ **do**
  $n_\star^k \longleftarrow$ attack to be most probable (Equation 3);
  Select $\phi^{k+1} \in \Gamma^+|$ so that after executed, $n_\star^k$ is still maximum (Equation 7);
  Execute ϕ^{k+1};
  $\Gamma^- \longleftarrow \Gamma^- \cup \phi^{k+1}$;
  Update N_x^{k+1} (Equation 4);
end

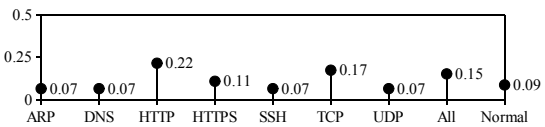
$$\phi^{k+1} = \arg\max_j \{p(N_x^{k+1} = n_\star^k|\Gamma^-, \phi_j)\}_{j \in \Gamma^+} \tag{7}$$

This criterion analyzes the next possible posterior probabilities $p(N_x^{k+1} = n_\star^k|\Gamma^-, \phi_j) = p(N_x^{k+1}|\Gamma^-) p(\phi_j|N_x^{k+1} = n_\star^k)$ that could happen. Note that we do not depend on the outcome of the probe (true or false). We loop for the complete set of remaining probes Γ^+ and select the one that maximizes the prediction 7 of the new posterior. This is somehow intuitive, because we seek for an affirmation of the current attack estimate that results in an increase in posterior probability in order to exceed the threshold in Equation 6. Since the remaining probes are not designed for testing the current attack estimate, the posterior only increases if the new probe also holds some information about the current attack (encoded in its CPT), although it favors a different one.

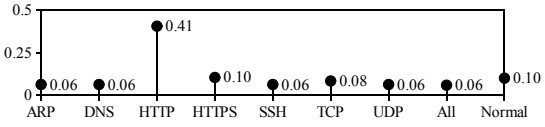
We map the above reasoning into the Algorithm 1 and we show in Fig. 2 a practical example of the updating process after each iteration when we deployed the dropping of HTTP packets attack.



a) Starting vector. We use a equal distributed prior probability.



b) Updated state after launching the probe ϕ_3. HTTP has highest probability.



c) Final state after launching probe ϕ_6. The node is dropping HTTP packets.

Figure 2. Sequence of the updated state values when detecting the dropping of HTTP packets.

Table II
CONDITIONAL PROBABILITY TABLE $p(\phi_1|N)$

ϕ_1	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
true	0.99	0.20	0.50	0.30	0.40	0.30	0.50	0.50	0.40
false	0.01	0.80	0.50	0.70	0.60	0.70	0.50	0.50	0.60

C. Optimal Probe Selection

To answer the question of which order of probe execution is most suitable, we define an optimization criterion that is based on the current posterior and a prediction step. Note that we allow each probe to be executed only once. This leads to a recursive selection mechanism that is optimal with respect to the optimization criterion. The criterion has to be evaluated anew at each time step and results in the next best probe to be executed. To answer the question on the minimal number of probes to execute, we define a simple stopping criterion that is based on a threshold on the maximum of the posterior. Once the maximum posterior probability is higher than the threshold, the AP-NIDS stops probe execution and provides the final classification result.

The selection of the first probe. Special care has to be taken in the selection of the first probe since the choice of the first probe can have a huge impact on the amount of probes to be launched afterward. Initializations of this kind should involve as much prior knowledge as possible/available.

In our experiments, the system administrator determines the first probe to be launched. Here the administrator's experience is needed—the administrator is assumed to know which attack is more common to occur in the network where the AP-NIDS is deployed. Practical alternatives to choosing the first probe could be:

- Selecting very critical probes first: the probe that is designed to detect the attack that is most harmful for the actual deployment of the network.
- Choosing very certain probes first: the probe offering a maximal probability in the CPTs $p(\phi_i|N_x)$ for one specific attack.
- Choosing more likely probes first: the probe that maximizes the prior probability $p(N_x^0)$.

Of course, a tradeoff between all three criteria is also possible. A probe that is 1) quite critical, 2) quite certain to detect and 3) quite likely to occur.

Implementation of the CPTs. The CPTs are build on statistical analysis of data, or/and a logical reasoning subjective to the administrator of the AP-NIDS. How to build the CPTs is out of the scope of this paper, and the CPTs we use for the evaluation of this paper are available to download together with the DogoIDS source code. As an example, to perform our experimentation we configured our AP-NIDS with the set of probes described in Table III, and the CPT associated to the outcome of ϕ_1 is given in Table II. The columns in Table II correspond to the different possible classes or attacks: n_1 for "dropping of ARP packets", n_2 for "dropping of DNS packets", ..., n_9 for "normal (no dropping)".

Table III
PROBES IMPLEMENTED FOR DETECTING SELECTIVE DROPPING OF DIFFERENT SERVICES.

Probe	Reference	Type of packets dropped	State
ϕ_1	ARP	ARP Requests and Responses	n_1
ϕ_2	DNS	UDP packets with port 53	n_2
ϕ_3	HTTP	TCP packets with port 80	n_3
ϕ_4	HTTPS	TCP packets with port 443	n_4
ϕ_5	SSH	Every packet with port 22	n_5
ϕ_6	TCP	Every TCP packet	n_6
ϕ_7	UDP	Every UDP packet	n_7
ϕ_8	All	Every UDP and TCP packet	n_8
	—	Normal condition (no dropping)	n_9

V. EXPERIMENTAL DESIGN

We detail the experimental design for evaluating our Bayes classifier.

A. Goals

Bayes classifier (attack detection and classification). The goal is to experimentally demonstrate that using our proposed Bayes classifier and probe selection algorithm have notable advantages in comparison to running the complete set of probes. First, we want to examine how the number of probes executed to detect a particular state decreases when using our classifier. Consequently, we expect that the overall detection time also decreases.

Overhead. We analyze the effects on the network overhead. We examine if employing an AP-NIDS highly loads the network with probes, and how it is decreased with our Bayes classifier. We also examine if varying the throughput of the network harms the active probing.

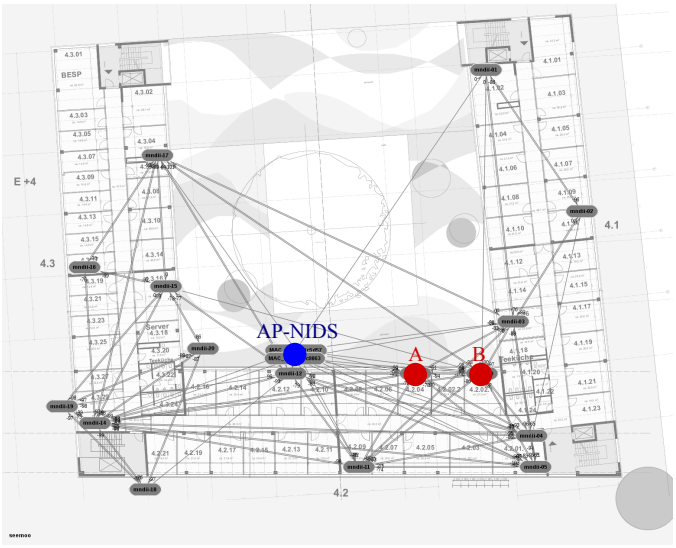


Figure 3. Our indoor mesh testbed and a snapshot of the network topology. Note that the location of the AP-NIDS nodes does not represent their exact physical location (they are located inside the room left of Node A).

Table IV
SETTING OF OUR EXPERIMENTS.

Parameter	Value/Description
Mesh network	
Number of nodes	16 fixed nodes + 2 netbooks (as AP-NIDS)
HW fixed nodes	PC Engines series Alix 3D2 boards [17]
HW notebooks	Asus Eee-Pc 1005HA
Nodes OS	Ubuntu 12 (netbooks) and Debian 5 (fixed nodes), with the wireless-testing kernel 3.5.0-rc4 [18]
Mesh protocol	IEEE 802.11s implementation from open80211s [18], using the Atheros drivers ath5k and ath9k
TX power	20 dBm
Data rate	Auto, up to 54 Mbit/s (Minstrel algorithm)
RTS/CTS	Off
AP-NIDS setup	
IDS node	2 netbooks (for emulating 2 interfaces)
Target node	Node B (see Fig. 3)
Background traffic	
Traffic tool	iperf
Traffic type	UDP packets, CBR
Server node	Node B
Client node	Node A
Traffic throughput	0 (0%), 6.25 (25%), 12.5 (50%), 18.75 (75%), 25 (100%) Mbit/s

B. Setup

The most relevant settings of our experiments are described in Table IV.

AP-NIDS. We perform the evaluation of this paper using our open source proof-of-concept DogoIDS. It is written in Python and runs under Linux. We implemented our Bayes classifier and configured the rules of Table III (for the states n_1–n_9). We employ 2 nodes running as an AP-NIDS. One node runs the AP-NIDS, and the other node works as final destination (MAC address) for the testing packets. The second node actually emulates having a second interface on the first one. Using two interfaces for an AP-NIDS is an implementation criterion and not a requirement [4].

Bayes engine. The conditional probability tables utilized in our evaluation are depicted in Table V. We build the CPTs using logical reasoning. For example, we give high values when the probe results are true for the attack they are supposed to be designed for. Note that the values of the rows are normalized before being combined with older measurements (for comparison, see in Fig. 2 the vectors already normalized). We set the probe ϕ_3 (dropping of HTTP packets) as the starting probe. This probe give us enough information about other attacks and it does not launch many probes (2 probes is not much in comparison to, for example, 5 of the dropping of HTTP packets).

Testbed. We employ an indoor testbed composed of 16 mesh nodes installed on two floors of an office building. We use the IEEE 802.11s mesh protocol. Figure 3 depicts the setup of our testbed and a snapshot of the network topology.

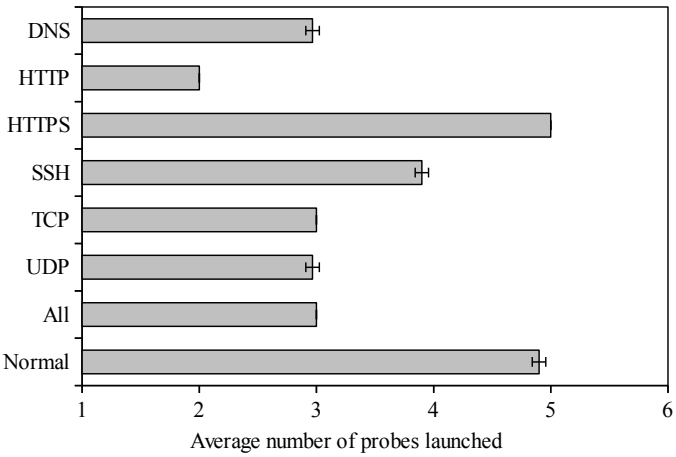


Figure 4. Number of probes needed to be launched for detecting different attacks.

Regarding the background traffic, we set 25 Mbit/s as the maximum achievable throughput between two nodes, and we employ different percentages of it for varying the load. We refer to 0 Mbit/s when we do not employ background traffic. We perform 50 analyses per experiment, and 3 replications of each test. The error bars in the figures represent the standard deviation.

Attacks. To perform the evaluation of this paper, we use *selective dropping of packets*. We find this attack of high interest for a couple of reasons: as explained in [1], this attack can severely degrade the performance of the network if, for example, a small number of highly critical control packets are dropped; in addition, it allows to define several detection rules for different services which helps us evaluating the Bayes classifier. We implement the attacks of Table III (with the exception of the dropping of ARP) by modifying the module `mac80211` of the wireless-testing Linux kernel. In particular, we modify some of the functions of the file *net/mac80211/rx.c* in order to allow the dropping of packets according to their transport protocol (TCP and/or UDP) and port number. This is necessary because in IEEE 802.11s the forwarding of frames is performed at layer 2 and they do not go up in the stack, so we cannot use layer 3 filters such as iptables. We remark here that the attack is relatively easy to implement if the attacker has basic Linux knowledge; easy but yet harmful attacks are usually preferred by attackers.

VI. RESULTS

In this section we present and we analyze the results of our experiments.

A. Number of Probes Launched and Detection Times

The goal is to observe how the inference model reduces the number of probes executed. As shown in Fig. 4, launching all the probes to detect an attack was not required in any of the analyzed cases. Even when the node was not dropping packets (Normal), only 5 of the 8 probes implemented were launched. In the best case, for detecting dropping of HTTP packets, only two probes were necessary, which represents

Table V
CONDITIONAL PROBABILITY TABLES IMPLEMENTED FOR OUR EXPERIMENTATION

CPT	ϕ_i	n_1 (ARP)	n_2 (DNS)	n_3 (HTTP)	n_4 (HTTPS)	n_5 (SSH)	n_6 (TCP)	n_7 (UDP)	n_8 (All)	n_9 (Normal)
$p(\phi_1\|N)$	true	0.99	0.20	0.50	0.30	0.40	0.30	0.50	0.50	0.40
	false	0.01	0.80	0.50	0.70	0.60	0.70	0.50	0.50	0.60
$p(\phi_2\|N)$	true	0.20	0.90	0.50	0.30	0.30	0.30	0.90	0.50	0.40
	false	0.80	0.10	0.50	0.70	0.70	0.70	0.10	0.50	0.60
$p(\phi_3\|N)$	true	0.70	0.30	0.99	0.50	0.30	0.80	0.30	0.70	0.40
	false	0.30	0.70	0.01	0.50	0.70	0.20	0.70	0.30	0.60
$p(\phi_4\|N)$	true	0.60	0.60	0.60	0.99	0.60	0.60	0.60	0.60	0.01
	false	0.40	0.40	0.40	0.01	0.40	0.40	0.40	0.40	0.99
$p(\phi_5\|N)$	true	0.50	0.40	0.50	0.40	0.99	0.50	0.50	0.50	0.30
	false	0.50	0.60	0.50	0.60	0.01	0.50	0.50	0.50	0.70
$p(\phi_6\|N)$	true	0.50	0.50	0.01	0.50	0.50	0.75	0.50	0.80	0.40
	false	0.50	0.50	0.99	0.50	0.50	0.25	0.50	0.20	0.60
$p(\phi_7\|N)$	true	0.50	0.30	0.50	0.50	0.50	0.30	0.99	0.80	0.40
	false	0.50	0.70	0.50	0.50	0.50	0.70	0.01	0.20	0.60
$p(\phi_8\|N)$	true	0.50	0.50	0.50	0.50	0.50	0.30	0.40	0.99	0.40
	false	0.50	0.50	0.50	0.50	0.50	0.70	0.60	0.01	0.60

25% of the set of probes. If the AP-NIDS would not have an algorithm for selecting the probes, it would need to transmit the complete set of probes to detect the attack precisely, given that the success of one probe is not enough to alert of the attack with confidence.

In Fig. 5 we show the average duration of the execution of each probe. These times include the transmission of the probe plus the gathering of data afterward. Intermediate times such as pauses between probes and processing times are not considered because they do not belong to the individual probes but to the analyses. As we can observe, the duration of the probes is different for each probe. Some probes demand about 11 s, while others around 30 s. If we compute the average of all the probes, we get a probe duration of 21.42 s $\pm$1.53. It is easy to observe the time saved when using the probe selection algorithm. For each probe that is not executed, the analysis requires 21.42 s less in average. This is an important reduction if we consider that the AP-NIDS is set with many different probes.

Figure 6 depicts the average duration of the complete active probing process. As we already observed in Fig. 5, some attacks take more time to be detected because more probes are launched (the analysis consist of several probes). The duration of the active probing is slightly affected by the traffic load of the network. In Table VI, we show the analysis duration for detecting the dropping of SSH packets, and for detecting the dropping of UDP packets when the network is loaded with different levels of traffic. Only when no background traffic is present, 0 Mbit/s, the analysis duration is in all cases shorter, because less data has to be processed by the IDS (it searches for the testing packets in the traffic gathered.)

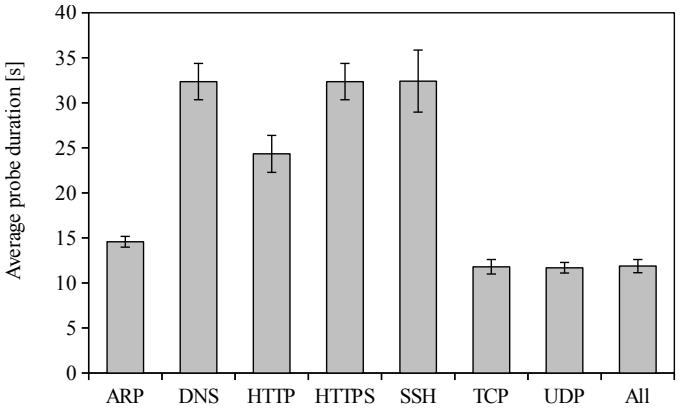


Figure 5. Average total duration of the individual probes.

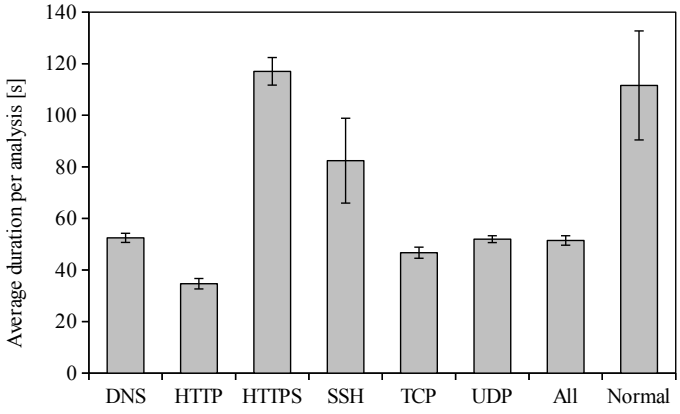


Figure 6. Average total duration of the detection of the different attacks.

Table VI
AVERAGE ANALYSIS DURATION FOR DIFFERENT BACKGROUND TRAFFIC LEVELS

Background traffic	Analysis duration [s]
Attack launched: dropping of SSH packets	
0 Mbit/s	87.76 ± 6.83
6.25 Mbit/s	101.84 ± 26.33
12.5 Mbit/s	109.54 ± 15.07
18.75 Mbit/s	101.74 ± 28.52
25 Mbit/s	107.01 ± 28.57
Attack launched: dropping of UDP packets	
0 Mbit/s	51.32 ± 3.80
6.25 Mbit/s	65.55 ± 11.38
12.5 Mbit/s	66.66 ± 10.20
18.75 Mbit/s	61.28 ± 8.96
25 Mbit/s	64.54 ± 6.52

Table VII
RESULTS OF THE ATTACK DETECTION. SSH AND UDP ARE ADDITIONALLY MEASURED WITH VARYING BACKGROUND TRAFFIC.

Attack	True positives	Well-classified	Misclassified
DNS	100%	93%	7%
HTTP	100%	100%	0%
HTTPS	100%	100%	0%
SSH	100%	93%	7%
TCP	100%	100%	0%
UDP	100%	97%	3%
All	100%	100%	0%
Normal	93%	93%	7%
Attack	**True positives**	**Well-classified**	**Misclassified**
Dropping of SSH packets			
6.25 Mbit/s	100%	83%	17%
12.5 Mbit/s	100%	93%	7%
18.75 Mbit/s	100%	80%	20%
25 Mbit/s	100%	90%	10%
Dropping of UDP packets			
6.25 Mbit/s	100%	97%	3%
12.5 Mbit/s	100%	100%	0%
18.75 Mbit/s	100%	93%	7%
25 Mbit/s	100%	100%	0%

B. Attack Detection

Our goal is to show if our Bayes classifier detects and labels correctly the different attacks we have implemented. We consider the results for the detection of attacks to be quite satisfactory. We depict the results in Table VII. We define as *true positive* an attack that takes place and it is detected (prior to the classification, provided that it is classified as an attack) by the AP-NIDS. A *well-classified* attack is an attack that takes place (true positive) and it is correctly classified by the Bayesian engine. On the contrary, a *misclassified* attack is an attack that takes place but it is not correctly classified, although it is still as an attack identified (for example, a dropping of TCP packets classified as dropping of HTTP packets). We differentiate between the true positives and well-classified attacks for the following reason: to validate our Bayes classifier, the number of well-classified attacks is an important metric to examine how good/bad it classifies the attacks. However, for intrusion detection, it is more important to firstly detect that an attack takes place. Ultimately, if the attack is detected but misclassified, an alarm will still be triggered and countermeasure actions can be taken. According to the results in Table VII, we observe that our Bayes classifier had some minimal troubles in classifying some attacks, but in any case the well-classified attack goes under 93% in the absence of background traffic.

Additionally, we are also interested in observing what happens if there is background traffic present. In Table VII we also depict the results for varying background traffic when detecting the attacks dropping of SSH packets, and dropping of UDP packets. The background traffic does affect the attack detection. In the worst case, we observe that the dropping of SSH was 80% well-classified. The reason of the misclassification is as follows. For detecting, for example, the dropping of SSH packets, the AP-NIDS launches four different probes (see Fig. 4). If one of these probes gives an incorrect result (a false detection, i.e., "true" when it should be "false", or vice versa), the Bayes classifier gets a wrong belief about the probability of one of the possible states, and it unleashes a set of wrong decisions of the probe selecting algorithm and, eventually, a wrong inference about the attack. For this reason, improving the attack classification implies improving the "robustness" of the individual probes. This is, however, not a drawback of the Bayes classifier. If a probe fails, even when transmitting the complete set of probes, the IDS can get an erroneous detection.

C. Overhead

Our goal is to measure the impact of the active probing in the network overhead. DogoIDS transmitted in the maximum case 1.85 Kbit/s for a complete analysis. That was the case of HTTP, including the control packets and testing packets generated. We argue that an AP-NIDS does not create a harmful amount of overhead. For example, our testbed handles 25 Mbit/s in the best case. The throughput generated by DogoIDS represents only 0.01% of the maximal achievable throughput. For a 25% utilization (6.25 Mbit/s), the throughput of DogoIDS represents 0.03%. Of course, the amount of data generated by DogoIDS depends on how many probes are executed and how many bytes are injected per probe. The data generated by the probe is a "user-defined" parameter. For example, in our experiments we generate "lightweight" testing packets, which contains a very small payload. We consider this is an appropriate choice but, at the end, the administrator of an AP-NIDS is free to create his/her testing packets as he/she considers. However, regardless of the individual testing packets, with our Bayes classifier and probe selector we directly reduce the number of injected bytes by reducing the number of probes launches, as discussed in Section VI-A.

On limiting the probe set to the available bandwidth. We claim that an AP-NIDS does hardly harm the network overhead. However, if this would be the case for very specific applications (bandwidth-limited sensor networks, for example), we propose to give a different view on the probe selection. An interesting approach is to limit the set of probes or attacks to detect a certain allowed throughput. For example, if network policies establish that only 5% of the maximal network throughput can be used for active probing intrusion detection, we can configure our AP-NIDS not to exceed that limit. That can be done by limiting the probe set, reducing the length of the testing packets, or increasing the intervals between probes or testing packets. We did not implement this idea but we leave it as future work.

VII. Conclusions

In this paper we highlight two contributions. We modeled a temporal-selective Bayes classifier to infer the state of a device under test. Different to similar classifiers proposed in the literature, the state of a device is a vector of all possible malicious behaviors. The transmission of probes feeds the Bayes classifier and updates the state. A recursive probe selection scheme, based on a prediction step, facilitates to reduce the number of probes while maximizing the insights gained. We modeled a general classifier and applied it to solve a particular problem, the malicious state inference for an active-probing-based network intrusion detection system for wireless multihop networks. We implemented our model and performed testbed experimentation. We showed how an inference model can improve the reduce the number of probes executed and therefore reduce overhead and detection time.

Acknowledgments

This work was gratefully supported by the German Research Foundation (DFG) within the GRK 1362 (www.gkmm.tu-darmstadt.de) and CASED (www.cased.de).

References

[1] T. Shu and M. Krunz, "Detection of malicious packet dropping in wireless ad hoc networks based on privacy-preserving public auditing," in *Proceedings of the 15th ACM conference on Security and Privacy in Wireless and Mobile Networks (WiSec '12)*. ACM, 2012.

[2] R. V. Boppana and X. Su, "On the effectiveness of monitoring for intrusion detection in mobile ad hoc networks," *Transactions on Mobile Computing, IEEE*, Aug 2011.

[3] R. do Carmo and M. Hollick, "DogoIDS: a mobile and active intrusion detection system for IEEE 802.11s wireless mesh networks," in *Proceedings of the 2nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy (HotWiSec '13)*. ACM, 2013.

[4] R. do Carmo and M. Hollick, "Analyzing active probing for practical intrusion detection in wireless multihop networks," in *Proceedings of the 11th IEEE/IFIP Annual Conference on Wireless On-demand Network Systems and Services (WONS '14)*. IEEE/IFIP, 2014.

[5] I. Rish, M. Brodie, N. Odintsova, M. Sheng, and G. Grabarnik, "Real-time problem determination in distributed systems using active probing," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2004.

[6] M. Brodie, I. Rish, and S. Ma, "Intelligent probing: a cost-effective approach to fault diagnosis in computer networks," *IBM Syst. J.*, vol. 41, no. 3, Jul. 2002.

[7] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom '00)*. ACM, 2000.

[8] Y. Huang and W. Lee, "A cooperative intrusion detection system for ad hoc networks," in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks (SASN '03)*. ACM, 2003.

[9] C. Tseng, S. Wang, C. Ko, and K. Levitt, "DEMEM: Distributed evidence-driven message exchange intrusion detection model for manet," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, D. Zamboni and C. Kruegel, Eds. Springer Berlin-Heidelberg, 2006, vol. 4219.

[10] O. Kachirski and R. Guha, "Effective intrusion detection using multiple sensors in wireless ad hoc networks," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*. IEEE, 2003.

[11] G. Vigna, S. Gwalani, K. Srinivasan, E. M. Belding-Royer, and R. A. Kemmerer, "An intrusion detection tool for AODV-based ad hoc wireless networks," in *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04)*. IEEE, 2004.

[12] F. Anjum and R. Talpade, "LiPaD: lightweight packet drop detection for ad hoc networks," in *Proceedings of the 60th IEEE Vehicular Technology Conference (VTC Fall 2004)*. IEEE, Sep 2004.

[13] F. Hugelshofer, P. Smith, D. Hutchison, and N. J. P. Race, "OpenLIDS: A lightweight intrusion detection system for wireless mesh networks," in *Proceedings of the 15th annual international conference on Mobile computing and networking (MobiCom '09)*. ACM, 2009.

[14] C. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. N. Levitt, "A specification-based intrusion detection system for AODV," in *Proceedings of the 1st ACM Workshop on Security of ad hoc and Sensor Networks (SASN '03)*. ACM, 2003.

[15] F. Oliviero and S. P. Romano, "A reputation-based metric for secure routing in wireless mesh networks," in *Proceedings of the Global Communications Conference 2008 (GLOBECOM '08)*. IEEE, 2008.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[17] "Alix Board 3D2 Wiki," online, http://goo.gl/fTzhO, Last accessed on June 11, 2014.

[18] "open80211s project," online, http://open80211s.org, Last accessed on June 11, 2014.