

# A wireless distributed intrusion detection system and a new attack model

Marco Domenico Aime    Giorgio Calandriello    Antonio Lioy  
Politecnico di Torino  
Dip. Automatica e Informatica  
10129 Torino - Italy  
{m.aime, giorgio.calandriello, lioy}@polito.it

## Abstract

*Denial-of-Service attacks, and jamming in particular, are a threat to wireless networks because they are at the same time easy to mount and difficult to detect and stop. We propose a distributed intrusion detection system in which each node monitors the traffic flow on the network and collects relevant statistics about it. By combining each node's view we are able to tell if (and which type of) an attack happened or if the channel is just saturated. However, this system opens the possibility for misuse. We discuss the impact of the misuse on the system and the best strategies for each actor.*

## 1 Introduction

Denial-of-Service attacks are a threat to informative systems which aim at blocking the service to legitimate users, which is often the first step in a series of attacks. In a shared medium environment an attacker could block another user to get more bandwidth, or a service provider could be attacked by a competitor to lower its reputation in front of the customers.

In wireless networks, DoS attacks can be categorized according to the features they exploit. We can distinguish between attacks to the authentication mechanisms (*Deauthentication flooding*, *EAP-Logoff*), to the power saving capabilities (*Sleep deprivation*) and to the radio layer (*jamming*, that is the emission of power over a frequency to disturb communications [7]). A number of them can be easily implemented with commodity hardware [4]. The problem with wireless networks is that it is very hard, if not impossible, to distinguish between "normal" malfunctioning (for example, random interferences on the channel or temporary saturation of the Access Point) and attacks.

Intrusion detection systems (IDS) on wireless networks are an open research topic. In the wireless world, *anomaly* detection seems to fit better as many attacks happen at MAC

level, which means that it's very difficult to isolate fixed sequences of packets to use in a *signature*-based IDS. Also, as wireless networks are a relatively new technology, it's likely that new attack methods will be found [3, 8]. What also emerged, is that detection should be distributed, as some attacks (and jamming in particular) cannot be detected by a single node [7].

In this paper we propose an attack detection mechanism based on shared monitoring of the network by all nodes, which should be able to tell whether the network is experiencing a real malfunctioning or is under attack. Such a system in the hands of service provider can give it a good way to prevent attacks, and it can be used in a public service context as well, by allowing clients to monitor the Quality of Service (QoS) and report when it's not achieved. On the other side, the operator can advertise a better service as it gives its clients a way to control it. Actually, it lets us define incentives mechanism which force the operator to guarantee high QoS. This system can however be misused by sending false attack reports, so we model the impact of liars and discuss the stability of the whole system. The use of "economic models" and incentives is of increasing interest in networking research [6], while the impact of a cheating and bandwidth-greedy user is discussed in [5].

In the discussion we reach different goals: we present a model of a public WiFi system which includes attack detection mechanisms, we analyze its evolution under the presence of cheaters and we present benefits from the distributed detection mechanism.

## 2 IDS for public WiFi system

In our scenario, WiFi for public service is offered by an operator to some users. Users pay a subscription cost and are offered a service with a given quality. The actors of the system are as follows.

**Operator** The WiFi operator, or service provider, runs the network infrastructure. It sets the QoS level and its

goal is to maximize its income. This can be achieved through different mechanisms: for example, by under-sizing the network to save on maintenance costs, or by keeping service to a good level as happier users are better customers than unhappy ones. Its best strategy in different scenarios will be outlined in the following of the paper.

**Cooperative users** This category includes users interested in the service only: these users will take the service “as is”.

**Unhappy/uncooperative users** Users under this category are unhappy about the current service and they will complain about it. They will perform all possible actions to get a better service or to get a refund. Given the required expertise and power to “make complaints” about the service, a cooperative user can become unhappy depending on the quality of service provided, and vice versa. The relation between the service level and the percentage of unhappy users in the system will be presented later in the paper.

**Cheaters** Cheaters are users who, regardless of the QoS, will try every action to increase their revenue. They will still benefit from the service itself, but will also try to “hack” it.

## 2.1 Distributed detection

We describe now the mechanism for the distributed detection system. The basic idea is to set up a *monitor* at each node in the network, as already described in [5], to produce *evidences*, and then share them among all nodes. An *evidence* a set of relevant information about the network state.

The monitor can be thought as an instance of the Ethernet [1] network packet sniffer: it captures traffic and displays detailed information on it. For each captured packet, Ethernet displays a complete view of the packet headers (from Ethernet to application level) and payload, and adds some general statistics as a timestamp, frame number and length in bytes. For our purposes we’ll look at the Ethernet level header, and as we’re focusing on 802.11 frames we’ll consider source, destination and BSSId addresses, sequence number, frame type and subtype and the Retry flag. Together with the captured packets, we add relevant statistics collected by the device driver, like counters for transmission retries and for frames received with wrong FCS (other papers [7] use different statistics as signal strength and carrier sensing time), and packet transmission time. We build in this way a list of *events* at each node. Events are the single transmitted packets or the times in which the channel is idle, which can be inferred from the timestamp of the packets and the packet transmission times.

The combination of different lists of events leads to better understanding of what happened in the network, in particular in distinguishing jamming attacks and channel failures, where packets are sent by one peer and never received by the other peer (a more detailed analysis of jamming effects and their detection can be found in [2]). Both a channel failure and a jamming attack make the FCS check of a packet fail, thus the packet in transit will be incorrectly received and dropped, incrementing the “dropped frames” counter in the device driver at the receiver. The difference between the 2 cases is the amount of incorrectly received frames at the receiver.

To exemplify this, let’s suppose to have the receiving station under a jamming attack. The jamming area can be thought as a border line where all packets crossing it become scrambled. The monitor placed at the source will see the frames sent on the channel, while the monitor placed at the receiving station won’t see anything received correctly and will see the counters for incorrectly received frames increase. The sender will retry the transmission a number of times, and this will be seen by the monitor placed there. All these retransmissions will be dropped as well, incrementing the counter again.

We are able to detect the attack by combining what both monitors saw, as a single one is not able to do the same: the receiver’s evidences (no packets received and counter updated) are in fact not enough to distinguish the attack. For the receiver, receiving incorrect frames can happen for various reasons: frames from stations at the limit of the radio range, frames from neighbor networks or noisy channel are all examples of this. If the counter is not updated, then staying idle without having transmissions aimed at itself or experiencing a device failure is undistinguished from being under attack. On the other side, the transmitter cannot tell if the other peer is out of range given the retransmissions only.

### 2.1.1 Matching the lists of events

The basic algorithm to match two lists of events is as follows: we start from the first list and for every event (packet or channel idle) we try to find a matching event on the second list that is, given a packet we look for it on the second list. As we don’t have cheaters into play for now, what we find is that for every packet on the first list we find it on the second one if the network worked fine, else we find a channel idle event if some problem (jamming or malfunctioning) happened. Continuing the example above, we’d have transmitted packets on the first event list and channel idle (together with a high number of dropped packets) on the second one. We can find unmatched events on the second list at the end (for example if the first node was jammed), so we swap the 2 lists and run the matching algorithm again. The final output is a single list of events which combines

the two. To tell the case we fall in, we can imagine to run a signature-based detection engine on the aggregated list.

Jamming and channel failure have the same basic signature (which is packets transmitted and never received), but differentiate on their position in the event list. A few packets disappearing here and there are index of channel failures, while a sequence of disappearing packets is considered as jamming. A large number of non-consecutive channel failures is index of bad QoS.

Since all nodes participate in the detection process, we extend it in order to match multiple lists. The idea is to merge one list at a time with the result of the previous merge. In other words, we merge lists #1 and #2, then we match the result with list #3, until we processed every list. We obtain in this way an aggregated list of all events which happened in the network in a given time frame.

We have to notice here that a node might not overhear the traffic of every other node because of range. We supposed that each node has relevant information to offer, but this is not always true. In the follow up of the paper we will consider all nodes to report relevant information to the aggregated list.

The key feature here is that the monitoring system is *distributed*. A single station alone cannot tell if it is experiencing an attack or just a temporary network failure, and cooperation among all nodes is required for the nodes to understand what is going on.

The event lists are shared among all nodes in the network. All nodes send their evidences to every other node in the network. For the scope of this paper we won't deal with details on the protocol used to share the lists, nor with their protection during transmission. If we experience a jamming attack we assume we'll be able to share evidences at a later time. We also suppose to have a secure authentication mechanism in place, to allow only authorized users to take part in the protocol.

Every node executes the matching algorithm to generate the aggregated event list to have a clear view of what happened in the network in the given time frame.

### 2.1.2 Commercial use

Another possible use of this mechanism is of commercial nature: the operator can use it to advertise a more secure and robust service (hardware failure is fixed more quickly as it's detected immediately, security breaches can be investigated sooner for the same reason), attracting more potential users.

To model and investigate this, we introduce the use of incentives. If the users monitor the channel and report that QoS is not respected the operator will pay back a *fee* to each affected user. The fee is a percentage of the subscription cost and is paid once a threshold of QoS violations have been reached.

## 2.2 System misuse

The system presented so far works as soon as cheaters stay out of the game. Why should a user cheat? The main reason is to get an advantage over other users. As stated in [5], nodes can alter their network card random backoff times and get an advantage over unmodified ones. In detail, a modified node will win the contention for the channel more often, getting a higher bandwidth share. Other techniques to do so are to launch DoS attacks against another nodes, like jamming or *Deauthentication*. Another possible reason would be to get the fee from the operator when the QoS is good in the commercial scenario we outlined above.

For the following of this paper, we'll consider this later case. Cheaters modify their lists of events to pretend to have bad QoS while the it is good to get the fee from the operator. We'll explain how to treat the other case later in the paper.

Cheaters will make the matching of the event list fail. In fact, a cheater will provide a list which is (at least in part) incompatible with the correct ones provided by the other honest nodes. For example, let's imagine that a node receives a packet X and claims not to have received it. The sender will of course report that it sent the packet. The receiver will alter his event list by marking packet X+1 from the sender as X, packet X+2 as X+1 and so on. When the matching will take place, it will show this difference.

We modify then our algorithm, and for every event we keep track of which node reported it and of clashing and incompatible events. In the example above, assuming A as the sender and B as the receiver, the list will report "channel free (reported by node B) AND packet X from A to B (reported by node A)" "packet X from A to B (B) AND packet X+1 from A to B (A)", "packet X+1 from A to B (B) AND packet X+2 from A to B (A)".

Under the hypotheses that all nodes are in range of each other, and that each node is either honest or cheater, when we try to build an aggregated list of events we'll end up with all honest users agreeing on a list, and cheaters disagreeing from it (eventually agreeing among themselves). What we are doing is building clusters from the different lists of events. Under an optimistic assumption that most nodes are honest we'll end up with a big cluster of honest nodes and a small number of outliers, representing the cheaters. However, if we don't assume the general goodwill of the users, cheaters can coordinate their attack and become the bigger cluster. In this case, since there are no trust mechanisms we cannot decide which cluster represents the honest users and which one the cheaters.

As we note that each node can trust only itself, we modify the matching algorithm: each node runs the basic 2-list matching algorithm between its own event list and each of the other nodes' lists. For each event, we mark if it's shared among the two nodes or not. At the end, the number of

matched events will be a measure of similarity between the two lists. When all the matchings will be done, each node will know how many other nodes share the same opinion as itself and thus how many other nodes are honest users or cheaters.

This system will just tell how many nodes agree or disagree with a given node. To make every node know the opinion of all the other nodes, each node repeats the matching algorithm using the list of events of another node (instead of its own) as starting point, and iterates on all nodes. This modified algorithm will require  $n - 1$  iterations to match a list of events with all the other ones. To match every list with all the other ones, if we do not repeat the already made matchings (for example, when matching node #1 with every other one we match #1 with #2, #3 etc. When we use node #2 as starting point we don't need to repeat #2-#1), then  $\binom{n}{2}$  matchings are needed, where  $n$  is the number of nodes.

The fact that nodes discover the opinion of other nodes is equivalent to running the algorithm in a centralized place. We assume that cheaters can cheat only on the event lists and not on the result of the algorithm.

### 2.2.1 Higher level modelling

At a higher level, we can think each node raising a vote about what happened in the network: *ok* when everything works, *bad QoS* when the network is saturated, *attack* when an attack is detected. Cooperative users will present their evidences which will support a certain conclusion, for example that the network is correctly operating, while cheaters will report that the network is *not* working by presenting forged evidences. We'll have two clashing and disagreeing views of what happened in the network - one group typically reporting *good* and another reporting *bad QoS*. To decide who is the "winner", we use the majority rule. This forces cheaters to cooperate to impose their view: a single cheater will never win.

This leads us to the question: how easy is to cheat? The easiest way is to have a peer generating an attack "on-demand" from outside the network: in this case we will have the cheaters supporting the view *bad QoS* while the cooperative users will support *attack*. This specific clash reveals this situation, so the operator can refuse to pay in this specific case. In case of an attack, the users will report *attack* and in this case the fee won't be paid, as the *bad QoS* level is not due to the will of the operator.

The other cheating case we presented, cheaters modifying their backoff times, can be reconducted to the former one. In fact, when cheaters manipulate their event lists they must cooperate, so there will be a number of fake *bad QoS* reports, while in the other case honest users will correctly report *bad QoS* as an effect of the cheating. The fee in this

STA \ AP	OK	Prob./Attack	Cheat
OK or Attack	C QoS	C QoS-ΔQ	C+S QoS-ΔQ
bad QoS	C-F QoS+F	C-F QoS-ΔQ+F	C+S-F QoS-ΔQ+F

Table 1. Table for the operator-users game

case represents the damage done to the operator, and the honest users will turn unhappy because of the cheating from malicious users.

## 3 Game theory modelling

To analyze the best strategies available to each actor we model the system as a game. Players are the operator and the users - seen as a global entity.

In this game, the operator has three choices: behave properly, network problems/attack (modelled as a separate strategy but not a free choice - the operator "chooses" this strategy in case of random network failures or in case of an external attack) and cheating (willingly undersizing the network). Cheating gives the operator the advantage to have more users than the network capacity, so to get more subscriptions. The users can report *good*, *attack* or *bad QoS*.

The game in tabular form is reported in Table 1. Its key is as follows:

**QoS:** Quality of Service: the quality of the service given by the operator

**C:** Subscription Cost: what operator get from each user

**ΔQ:** Quality of Service loss: quantification of how the service is of lower quality than standard

**S:** Operator saving: what the operator saves by cheating

**F:** User fee: what the users gain through cheating.

The costs in table 1 are computed as follows: if both operator and users behave correctly, the operator gets the subscription cost, while the user gets the QoS. If there are problems, or cheating from the operator, the user will always experience a service penalty. If the operator cheats and this is not reported (users choosing *attack* or *good*), the operator gets a reward, S, which represents its savings due to the inadequate infrastructure it deploys. If the users report *bad QoS*, they will always get the fee F, which in turn will be paid by the operator. Reporting *attack* or *good* leads to the same payoffs so they have been collapsed into a single row.

We can note that both players have a dominant strategy: the operator with *cheat* and the users with *bad QoS*.

This seems a doomed situation, however this model doesn't take into account the evolution of the population

in the system. Most of the users cheat only if the QoS is too low, and cheating from the operator leads to lower QoS and to more cheating from the users. We'll examine this in closer detail in next section.

## 4 System evolution

The operator sets the QoS. The users decide if to cheat or not. Since the users subscribe to the service we consider the QoS to be the independent variable, and we model the relation between QoS and percentage of unhappy users as linear. In the following example we'll use the function  $P_u = 1 - x$ .  $P_u$  is the probability of an user to be unhappy with the service, and  $x$  the QoS.  $P_u, x \in [0,1]$ .

Given this relation, the operator can lower the QoS down to  $\bar{x}$ , where  $P_u(\bar{x})=0.5$ . This because when the unhappy users are the majority they cheat and get the fee. Note that without any monitoring system the operator can select any QoS at will without paying any penalty. The users either leave or take the service "as is". This way, the distributed detection system limits the possibility of cheating by the operator.

If we consider to have a fixed number of cheaters in the system in addition to the unhappy users, the relation changes to:

$$P_{c+u} = 1 + (P_c - 1)x \quad (1)$$

where  $P_c$  is the (fixed) percentage of cheaters in the system and  $P_{c+u}$  the probability for an user to be cheating, either as cheater or as unhappy user. For  $P_c = 0$  we have the relation as above. Note that  $x, P_{c+u}, P_c \in [0,1]$ .

### 4.1 Modelling users' cheating effects

We consider now a coordinated attack, meaning that users will actively cooperate to subvert the system. The system is composed by multiple cells, and the cheaters decide which ones to subvert. We also consider the unhappy users as cheaters: in fact, even if for different reasons, they are the same from the point of view of the operator for the purpose of the single attack.

#### 4.1.1 Knapsack problem analysis

We suppose a system made of many cells, with a given population, and with variable distribution of population into cells, ranging from uniform to very narrow Gaussian. We suppose a percentage of cheaters varying from 10 to 40%. Cheaters coordinate their efforts: a single cell is assigned a value equal to its size (for example, a cell containing 60 users has a value of 60) and a weight equal to half of its size +1 (in the previous case, 31), to reflect that cheaters must

Parameter	Formula
revenue	$k_1 t$
costs	$k_2 t x + k_3$
fees	$k_4 y$
users	$t = k_5 x$

**Table 2.** Table including some useful equations

be the absolute majority in a cell to subvert it. This is an instance of the 0-1 knapsack problem, and results for its simulation are reported in Appendix A. The number reported as "score" is the sum of the values of all objects taken, in this case it's the total number of affected users to whom the operator will have to pay the fee.

What we can observe at first is that the results are independent from both the size of the cells and the number of population. From the results we obtain the following experimental relationship:

$$y = 2(P_{c+u}) \quad (2)$$

where  $y$  is the amount of cheating, expressed as percentage of the users who get the fee.  $P_{c+u} \in [0,0.4]$ , as this is the input range we used in the simulations.

#### 4.1.2 Evolution of user population

Under a cheating attack honest users are not immediately affected, as the operator might reduce its QoS as a consequence of the fees it has to pay. This way its network will be even more undersized, with more problems and more honest users becoming cheaters. However, this trend cannot go forever as the operator will at some point shut the network down, giving no more service nor more fees back.

We model the idea in this way: in our system we have a fixed percentage of cheaters,  $P_c$ , and a percentage of cheating users,  $P_{c+u}$  which depends on the QoS,  $x$ , as in equation 1. Equation 2 outlines the relation between the amount of cheating and the percentage of cheating users. Table 2 includes some useful equations that model the operator's revenue, costs and fees.  $t$  is the number of users,  $k_1, k_2, k_3, k_4$  and  $k_5$  are all constants of proportionality. Costs are proportional to the number of users and the QoS, plus a fixed value<sup>1</sup>.

We can now calculate the the operator's profit (gain) function,  $G$ , as  $G = income - costs - fees$ . Substituting the formulae, replacing  $t$  and  $y$  with their respective functions and using Equations 1 and 2 we obtain:

$$G = -k_2 k_5 x^2 + [k_1 k_5 + 2k_4(1 - P_c)]x - (k_3 + 2k_4) \quad (3)$$

<sup>1</sup>For example, the cost of governative licenses to provide the service

which is a quadratic relation between the gain and the QoS, aimed towards bottom.  $x \in [0,1]$ . Since the  $y$  axis represents the operator's gain, we can assume that the function's vertex represents a positive value of  $G$  and thus there are 2 intersections with the axis  $x$ . The function's maximum is

$$x = \frac{k_1 k_5 + 2k_4(1 - P_c)}{2k_2 k_5}$$

which is positive as all constants are positive. We can assume to have a profit when  $x=1$ . Also, when  $x=0$  we have a profit of  $G = -k_3 - 2k_4$ , thus a negative value. Given these characteristics of our function, we can say that the maximum is located either in our domain or at  $x_{max} = 1$ . From here we can see that the operator tends to keep the QoS at maximum as it maximizes  $G$ .

The cheaters' impact will make the QoS decrease, thus making the number of unhappy users rise and operator's gain decrease. If the gain approaches zero or goes below it, the operator can either shut the service down or operate in loss for some time to lower the number of unhappy users. The first choice is against cheaters' interests as well, as we said they're interested in the service as well and not only on the fees. The second choice can be kept for a little time only, as the cheating level should lower after a while as it usually comes from a small number of real cheaters and a higher number of unhappy users.

## 5 Conclusions and future work

In this paper we proposed an intrusion detection system for wireless network based on distributed collection of relevant information, and showed that it can also detect jamming attacks. We also suggested a commercial use of the system, in order to provide a better service to customers: however, this use allows cheaters to come into play. Anyway, their impact is limited: we showed that the operator cannot lower the quality of service under a certain threshold (as without such a system), otherwise unhappy users will take over and get a pay back. We also showed that cheating users cannot push too much, otherwise the system will go towards the total shutdown. We achieve two goals: we detect more attacks and force the operator to give a decent service. We allow cheaters to come into play, but their impact is self-limiting as a working network is needed for them to play.

One interesting scenario to analyze would be with cheaters who don't care about the service, thus don't stop cheating when QoS gets too low. This might be a sabotage attack from a rival provider to get more market shares. It would also be interesting to add trust and user reputation mechanisms to the system, to improve the matching algorithm.

$\alpha$	% cheaters	Score
0.1	0.1	194
0.1	0.2	395
0.1	0.3	594
0.1	0.4	788
0.5	0.1	195
0.5	0.2	388
0.5	0.3	580
0.5	0.4	771
1	0.1	190
1	0.2	380
1	0.3	566
1	0.4	752

**Table 3. Simulation results for cell takeover, 32 cells, 1000 users, 0-1 knapsack analysis**

## A Simulation results

In table 3 there are the simulation results for the 0-1 knapsack analysis. The table reports only the case of 32 cells and 1000 users, as we experienced that modifying these parameters doesn't significantly affect the score.  $\alpha$  is the parameter telling the shape of the users' distribution into cells: 0 is narrow Gaussian, becoming uniform distribution at 1.

## References

- [1] Ethereal: a network protocol analyzer. <http://www.ethereal.com>.
- [2] M. Aime and G. Calandriello. Distributed monitoring of wifi channel. Technical report, Politecnico di Torino, 2005.
- [3] Y. an Huang and W. Lee. A cooperative intrusion detection system for ad hoc networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 135–147, Fairfax (VA), USA, 2003.
- [4] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the 11th USENIX Security Symposium*, pages 15–28, Washington D.C, USA, 2003.
- [5] M. Raya, J.-P. Hubaux, and I. Aad. Domino: A system to detect greedy behavior in iee 802.11 hotspots. In *Proceedings of ACM MobiSys*, Boston (MA), USA, 2004.
- [6] N. B. Salem, J.-P. Hubaux, and M. Jakobsson. Reputation-based wi-fi deployment. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(3):69–81, 2005.
- [7] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 46–57, Urbana-Champaign (IL), USA, 2005.
- [8] Y. Zhang, W. Lee, and Y.-A. Huang. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, 9(5):545–556, 2003.