Detection of Selfish Manipulation of Carrier Sensing in 802.11 Networks

Konstantinos Pelechrinis, Guanhua Yan, Stephan Eidenbenz,
and Srikanth V. Krishnamurthy, *Senior Member, IEEE*

Abstract—Recently, tuning the clear channel assessment (CCA) threshold in conjunction with power control has been considered for improving the performance of WLANs. However, we show that, CCA tuning can be exploited by selfish nodes to obtain an unfair share of the available bandwidth. Specifically, a selfish entity can manipulate the CCA threshold to ignore ongoing transmissions; this increases the probability of accessing the medium and provides the entity a higher, unfair share of the bandwidth. We experiment on our 802.11 testbed to characterize the effects of CCA tuning on both isolated links and in 802.11 WLAN configurations. We focus on AP-client(s) configurations, proposing a novel approach to detect this misbehavior. A misbehaving client is unlikely to recognize low power receptions as legitimate packets; by intelligently sending low power probe messages, an AP can efficiently detect a misbehaving node. Our key contributions are: (a) We are the first to quantify the impact of selfish CCA tuning via extensive experimentation on various 802.11 configurations. (b) We propose a lightweight scheme for detecting selfish nodes that inappropriately increase their CCAs. (c) We extensively evaluate our system on our testbed; its accuracy is 95% while the false positive rate is less than 5%.

Index Terms—Wireless Networks, Carrier Sensing, Clear Channel Assessment threshold, Denial of Service, Selfish behavior, Experimentation, Analysis.

1 Introduction

It is well known that the distributed coordination function (DCF) of the IEEE 802.11 MAC protocol provides long term fairness to users that are in the proximity of one another and share the wireless medium [1]. Recently, there have been many approaches that advocate the joint tuning of the transmission power and the clear channel assessment (CCA) threshold to improve spatial reuse and thereby, the achievable capacity in a WLAN [2][3]. Tuning the CCA threshold opens the door for a new kind of selfish or malicious behavior. By increasing the CCA threshold, a "misbehaving" user[1] will cause the carrier sensing at the MAC layer to ignore the transmissions of other users with which it shares the medium. As a consequence, (a) it may initiate transmissions when other transmissions are in progress thereby increasing collisions and, (b) it will not freeze its back-off counter while other nodes are transmitting packets; as a consequence it is able to access the medium much more frequently than other users and thus, enjoy a higher unfair share of the bandwidth. Given these adverse effects, it is critical to understand and quantify the extent to which fairness suffers from CCA manipulations in 802.11 networks. For this reason, we first examine the effects of such behavior on isolated links. This provides a fundamental understanding of the behavioral nuances of carrier sensing on real hardware.

Having acquired the above basic understanding we focus on AP-client configurations and we further examine such scenarios in more detail. As our main contribution, we propose a novel approach for detecting misbehaving clients with high accuracy. There are two observations that drive our approach. **First,** a misbehaving node that increases its CCA threshold is likely to have a good "link" to the AP to begin with. If this is not the case increasing the CCA can compromise the connectivity of the node. **Second,** by increasing the CCA threshold towards gaining an unfair share of the throughput, the misbehaving node *implicitly* raises the bar with regards to the RSSI (Received Signal Strength Indicator) required for correct decoding. The receiver circuitry only tries to decode packets that are received with an RSSI that is higher than the CCA threshold.

Based on the above observations, we design the Carrier sensing Misbehavior Detection (CMD) system. The key insight, evident from the above observations, is that a node that has increased its CCA threshold is likely to ignore *low power transmissions* from the AP. Thus, by sending low power probes, the AP can potentially detect such nodes with high accuracy. In order to reduce the overhead that will be incurred due to such probes, CMD first identifies a *set of possible* badly behaving nodes. This set consists of those nodes that are enjoying a significantly higher share of the throughput than their counterparts that are within the same cell. The probe messages are then only sent to the members of this set. Note here that, under saturated conditions where this problem is likely to be most critical,

- *Konstantinos Pelechrinis is with the Department of Information Sciences of University of Pittsburgh.*
E-mail: kpele@pitt.edu.
- *Guanhua Yan is with the Los Alamos National Laboratories.*
E-mail: ghyan@lanl.gov.
- *Stephan Eidenbenz is with the Los Alamos National Laboratories.*
E-mail: eidenben@lanl.gov.
- *Srikanth V. Krishnamurthy is with the Department of Computer Science and Engineering of University of California, Riverside.*
E-mail: krish@cs.ucr.edu.

This work is supported in part by the US Army Research Office under the Multi-University Research Initiative (MURI) grants W911NF-07-1-0318 and the NSF NeTS:WN / Cyber trust grant 0721941.

1. We use the terms misbehaving, cheating, greedy and selfish interchangeably. We also use the terms user, node and client interchangeably.

this set naturally excludes nodes that are at the periphery of the cell or nodes with poor links.

In more detail our contributions in this paper are as follows:

- We experimentally characterize the operation of CSMA/CA on commodity wireless NICs, by examining the effects of CCA tuning on isolated links. We further quantify the impact of selfish CCA tuning on the overall network performance of a WLAN. This is the first study that quantifies the extent to which, fairness suffers if this functionality were to be used inappropriately.
- We design and implement CMD for detecting such misbehaving clients in WLANs. CMD consists of two subcomponents: (a) The Throughput Monitoring Module (TMM), which identifies a candidate set of possible misbehaving nodes and (b) The Low power Probing Module (LPM), which transmits the low power probes to effectively detect the *real* misbehaving nodes from among this candidate set. The implementation of CMD does not require any modifications to the IEEE 802.11 driver or firmware and can be implemented in the user space in its entirety.
- We analytically compute system parameters for CMD such that low *false positive* (wrongly classifying a well-behaved node) and *false negative* (not recognizing a misbehaving node) probabilities are achieved. We validate our analytical results through measurements.
- We perform extensive experiments to evaluate CMD on an indoor WLAN testbed, with various configurations. Our experiments show that CMD detects misbehaving nodes with extremely high accuracy (95 %) with a very low false positive rate (< 5 %).
- We discuss possible strategies to mitigate the impact of misbehaving nodes after detection. In particular we propose and experimentally evaluate a simple, yet seemingly promising technique.
  We provide guidelines for possible mitigation schemes. We further examine a technique based on transmission power control from the AP. Our experimental results show the applicability of the proposed approach.

Our work in perspective: Selfish behaviors that target 802.11 functionalities have been considered and addressed previously. In particular, there have been many efforts that try to overcome behaviors where greedy nodes manipulate the back-off timers with 802.11 [4][5][6][7][8][9]. While a misbehaving node can enjoy lower back-off times by manipulating the CCA threshold (fewer instances of freezing the back-off counter), we wish to point out that the two attacks are not the same. In particular, unlike the other attacks, tuning the CCA threshold is **protocol compliant**; the 802.11 standard [10] does not specify a value for the CCA threshold. In fact, different wireless network interface cards (NICs) have slightly different CCA thresholds. Although currently, tuning the CCA threshold is a functionality that these cards implement in the firmware, there are ongoing efforts towards enabling this functionality [11]. There have already been research efforts that advocate the tuning of this threshold for performance improvements [2][3]. In addition, GNU software defined radios [12][13] are expected to fully support 802.11 soon; such coexisting platforms that allow CCA tuning could be misused to pilfer a higher share of the throughput.

We only consider uplink traffic since one might expect that the APs, which are usually controlled by service providers, are unlikely or do not have the incentive to cheat; stated otherwise, it is unlikely that downlink traffic will be prone to such misbehaviors. Furthermore, the uplink traffic of a WLAN is not a negligible percentage of the total AP traffic anymore [14]. The increasing popularity of *p2p* applications result in the generation of a high proportion of uplink traffic in commercial hotspots.

The rest of the paper is organized as follows. In Section 2 we discuss relevant CSMA/CA functionalities in brief and related work. In Section 3 we describe our testbed at UC Riverside. Our measurements for understanding carrier sensing and CCA tuning on isolated links are presented in Section 4. Section 5 examines the effects of selfish CCA tuning on WLANs. In Section 6 we present the design and implementation of CMD; we analyze its performance in Section 7. In Section 8 we discuss the results of our evaluations of CMD. Possible mitigation schemes and miscellaneous issues are deliberated upon in Section 9. Our conclusions form Section 10.

2 Background and Related Work

In this section we provide a brief description of relevant CSMA/CA functions and describe related work.

2.1 Relevant 802.11 functions

802.11's access policy is based on CSMA/CA. Each user needs to sense the medium idle for a specified time prior to transmitting data [15]. Whenever the perceived power on the medium is higher than the CCA threshold, a node must defer its transmission and enter the backoff state. Upon reaching this state, a node initiates a back-off counter with a random value. For each time slot that the medium is free, the counter is decremented; for each time slot that the energy on the medium is higher than the CCA threshold, the value of the counter is left unchanged (or *frozen*). When the counter value is decremented to zero, the node senses the medium again. If the power on the medium is lower than the CCA threshold (medium is idle) it transmits its packet; otherwise, it re-enters the backoff state and the expected counter value is now doubled.

When a misbehaving node increases its CCA threshold, it can result in the following effects:

- It can now ignore those signals that it *senses*, but are lower than this *new increased* threshold. Therefore, many of the signals on the medium have now no effect on the transmission opportunities of the node.
- Other nodes that use the default CCA will sense the transmissions of the selfish node and will defer their own transmissions for longer periods.
- If a transmission of the misbehaving node is not successful, it will enter the backoff state. However, since its CCA threshold is increased, it is more likely that it will not have to *freeze* its backoff counter; this is a consequence of the node sensing the medium to be idle even if there are ongoing transmissions.

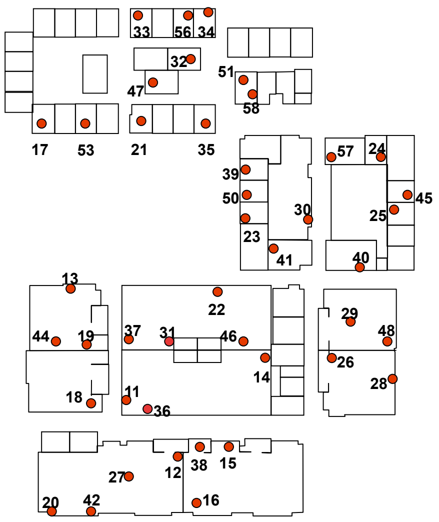


Fig. 1. The UCR wireless testbed.

2.2 Related studies

While there have been prior efforts on overcoming attacks that manipulate 802.11 functionalities, the attack considered in this paper has not received prior attention.

Attacks that violate the 802.11 back-off timers: Kyasanur and Vaidya [4] consider selfish behaviors where nodes deviate from the standard backoff mechanism of 802.11. They propose a mitigation scheme where the receiver explicitly assigns the backoff value to the sender. Konorski [9] proposes a misbehavior-resilient backoff mechanism. Gagalj *et al* [7] use game theory to develop a simple, localized and distributed protocol that guides multiple selfish nodes that deviate from the backoff mechanism to a Pareto-optimal Nash equilibrium. Radosavac *et al.*, [5] present a framework based on the Sequential Probability Ratio Test (SPRT) for detecting nodes that deviate from the backoff mechanism. Finally, Queseth [8] shows that it is hard to discourage selfishness by punishment if we cannot quickly detect these behaviors. All these studies however, are primarily related to the exploitation of the backoff mechanism, which is not the focus of our work. Note that in the considered setting, a node only increases its CCA threshold and does not directly violate the back-off policies; thus, these previously considered methods will not be effective.

Detecting other selfish behaviors: Raya *et al* [6] propose and implement DOMINO, a system for detecting various selfish behaviors in WLANs. DOMINO detects nodes that do not adhere to the standard backoff mechanism, send out data without waiting for the standard DIFS period, use an oversized NAV to retain the medium for a longer time, or intentionally corrupt frames to obtain unfair access to the wireless medium. In the attack considered, misbehaving nodes increase their CCA; none of the above behavioral trends are observed (as an example, the DIFS periods followed by the selfish nodes are legitimate). DOMINO cannot accurately detect an attack where nodes ”do not” freeze their back off counters due to ongoing transmissions. Consequently, DOMINO cannot detect possible CCA manipulations. Note that our approach can be complementary to DOMINO.

To the best of our knowledge we are the first to experimentally examine the selfish behaviors of nodes that try to increase their throughput by exploiting the CCA threshold functionality. Recently, Paul *et al* [16] proposed a passive monitoring technique to detect users that do not adhere to carrier sensing at all. They follow a centralized approach and require the deployment of separate monitoring nodes. Their evaluations through ns2 simulations illustrate the potential of the approach; the approach however in its current form relies on some simplified assumptions in its current form.

3 Experimental Setup

In this section we provide a brief description of our testbed and the experimental methodology that is followed.

3.1 Testbed Description

Our wireless testbed (Figure 1) is located on the 3rd floor of Engineering Building II at UC Riverside and consists of 42 Soekris net4826 nodes [17]; the nodes mount a Debian Linux distribution with kernel v2.6, over NFS. Each node is equipped with two miniPCI 802.11a/g WiFi cards, an *EMP-8602 6G* with Atheros chipset and an *Intel-2915*. We use the MadWifi driver [18] for the *EMP-8602 6G* cards. We use a proprietary version of the *ipw2200* AP and client driver/firmware of the *Intel-2915* card. With this version we are able to tune the CCA threshold parameter. Up to date information on our current testbed set up can be found in [19].

3.2 Experimental Methodology

The misbehaving clients exclusively use our Intel cards, since these cards allow CCA tuning. The default value for the CCA threshold is -80dBm. All nodes use the maximum power (18dBm). Our experiments span a large number of communications pairs (more than 80). In addition, we perform measurements with more than 2 clients associated with the same AP (up to 4 clients associated with a single AP), accounting for different levels of cross traffic. We conduct our experiments by mainly utilizing 802.11a, in order to be able to perform controlled experiments avoiding interference from co-located WLANs operating at 2.4 GHz. However, we have verified, that our findings hold with 802.11g as well (unless otherwise stated). We provide more details on every experiment in the following sections.

4 Effects of CCA Manipulation on an Isolated Link

Key to our study is understanding the effects of CCA tuning. The way that carrier sensing operates is crucial for both (a) inferring the effects of selfish exploitation of the CCA tuning mechanism, as well as, (b) designing an accurate detection scheme. We want to emphasize that providing a detailed understanding on the real world performance of carrier sensing is not the main focus of our work. An interesting experimental study, complementary to ours and in a different direction, of how carrier sensing works in practice can be found in [20].

For the purposes of our study we activate one link at a time and we progressively increase the CCA at its two end points, monitoring at the same time the achieved throughput. Figure 2 depicts the representative results for two links on our testbed. The remaining links exhibited the same qualitative performance.

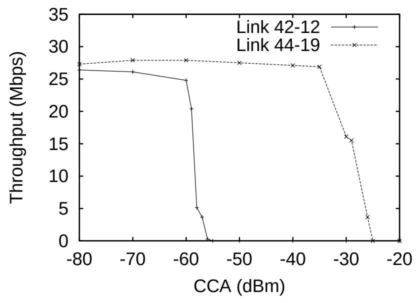


Fig. 2. Throughput as a function of CCA on isolated links.

We observe that slightly increasing the CCA, does not have any effect on the throughput enjoyed by the link; it remains practically constant. However, increasing the threshold above a specific value[2] - different for every link - results in a significant and sudden throughput degradation; eventually zero throughput is observed on the link as can be seen from Figure 2.

In order to understand the reasons behind this performance, we need to recall that most commodity cards set the receiver's sensitivity equal to the CCA threshold [2] [3] [21]. This means that at some point - when we keep increasing the CCA value - the RSSI at the receiver's circuitry will be lower than its increased CCA/sensitivity, and this will result in compromised connectivity; the receiver will ignore the packet and regard it as noise.

From these experimental results, we conclude that we cannot keep *blindly* increasing the CCA threshold on a link; we need to take the above effect into account when understanding the attack and designing a countermeasure. One could argue that we could increase the CCA only at the sender side, in order to avoid discarding the packets at the receiver. However, successful 802.11 communications involve MAC layer ACKs. As a result, even if only one side of the link increases its CCA threshold, a similar performance is observed. In fact, we have experimented with such scenarios and we present a sample result in Figure 3. Notice that the CCA_{cutoff} is different depending on which side of the link is using an increased CCA value. This is an artifact of the *asymmetry* on the wireless links. Using the terms $CCA_{cutoff-r_x}$ and $CCA_{cutoff-t_x}$ for the cases where the receiver or transmitter only, respectively, increases its threshold, we get:

$$CCA_{cutoff} = min(CCA_{cutoff-r_x}, CCA_{cutoff-t_x}) \qquad (1)$$

As mentioned earlier the CCA_{cutoff} value is different for every link and depends on **(i)** the environment, **(ii)** the distance spanned by the link, **(iii)** the transmission power and **(iv)** the propagation characteristics (e.g. frequency used). Figures 4(a) and 4(b) present the empirical statistics (CDF and PDF/histogram) of the CCA_{cutoff} for the links on our testbed, when the maximum power is employed and channel 60 (5.3 GHz) is used.

2. We refer to this value as CCA_{cutoff}.

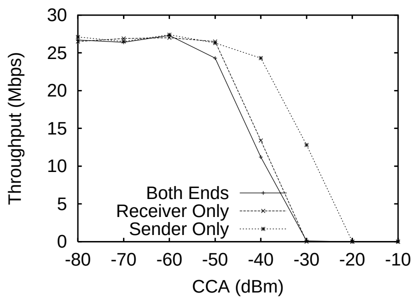


Fig. 3. Increasing the CCA on one end of the link only, has the same effect (link 22-31).

In the rest of this paper we focus on selfish behaviors considering uplink traffic in WLANs. The selfish client should not expect the AP to co-operate and increase its CCA too. As a result, our case study involves situations where only the sender, increases its carrier sensing threshold. The greedy entity needs to consider the trade off between ignoring ongoing transmissions and not being able to decode the MAC layer ACKs from the AP, when increasing its CCA. In the rest of the paper, we will focus on *aggressive* selfish strategies, that is, greedy users will use the maximum possible CCA that does not degrade the link quality to the AP. In Section 9 we will briefly examine the effects and the detection of less aggressive cheating strategies.

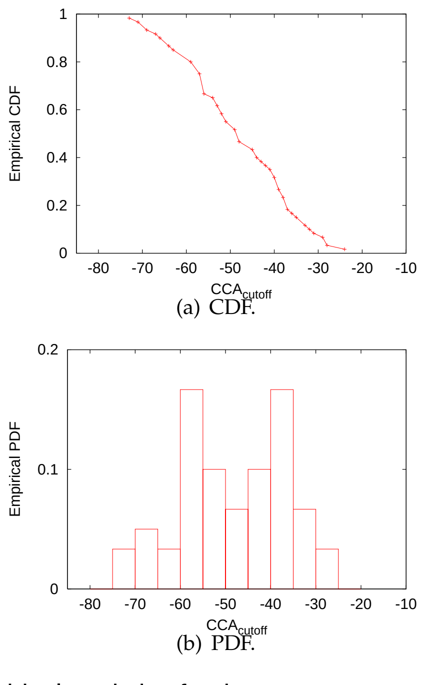


Fig. 4. Empirical statistics for the CCA_{cutoff}.

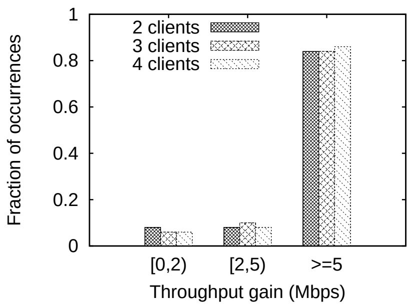


Fig. 5. Increasing CCA can be an effective greedy strategy.

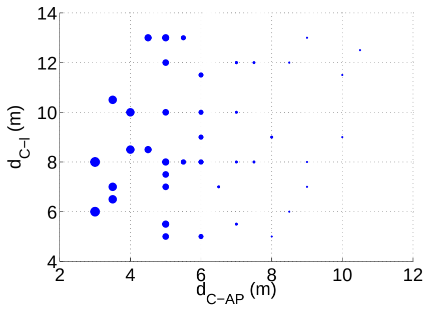


Fig. 6. Topology affects cheating gain.

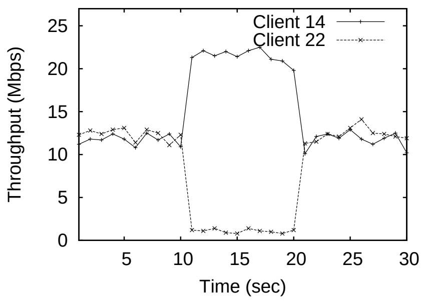


Fig. 7. Time trace for the case of UDP traffic.

5 Selfish CCA tuning on WLANs

The 802.11 MAC protocol, as discussed earlier, provides long term *max-min* fairness to nodes that share a link. Under saturated conditions all the nodes that share a link, essentially access the medium with the same probability. By increasing the CCA threshold, a node can pilfer a higher share of the medium than it is entitled to, from the other users. To reiterate, transmissions that arrive at the receiver circuitry with an RSSI lower than the CCA threshold are ignored. By increasing the threshold, a node can ignore a significant fraction of the transmissions that occupy the medium.

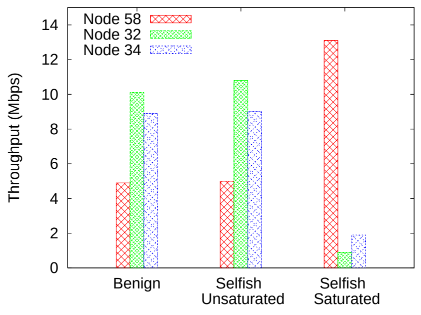


Fig. 8. A non-aggressive cheater is not harmful.

As described before, this not only causes an increased number of collisions but also allows the misbehaving node to reduce the fraction of the time that it spends in the back-off state.

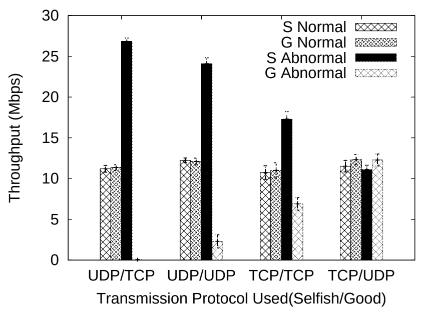


Fig. 9. Impact of different transport layer protocols (S:selfish, G:good user).

Our objective in this section is to study in more detail the effects of this greedy behavior in a WLAN setting. We experiment with various configurations (with varying locations of the APs and clients), different traffic patterns and various transport protocols, and we measure the throughput gains of the selfish clients relative to their fair share of throughput under normal operating conditions.

Experiments with saturated traffic: We depict our first results in Figure 5. The x-axis represents the throughput gains of the selfish clients and the y-axis represents the percentage of occurrences of this throughput gain (the gains are quantized into three levels); we vary the number of clients connected to the AP. We observe that in most cases (more than 85% of the 90 scenarios considered in total) the cheating user is able to gain significantly over the well-behaved clients affiliated with the same AP - at least a 5Mbps gain from its fair share is seen.

In some scenarios though (in fewer than 5% of the considered scenarios), the selfish client is unable to pilfer more than 2Mbps from the other clients. These cases arise when the selfish client is far from the AP (e.g., node 36 is the AP and node 22 is the selfish client) and as a result cannot increase its CCA to very high values; doing so would result in its disassociation from the AP. These studies suggest that a selfish node is likely to choose a location that is as close to the AP as possible[3].

In order to illustrate the effect of the cheater's position on the achievable throughput gains, we experiment with two clients associated with the AP. Figure 6 presents the throughput gain, as a function of the cheater's distance from the AP d_{c-AP}, and the cheater's distance from the legitimate client d_{c-l}. In this figure, the area covered from each point is proportional to the cheating gain observed. Thus, points that cover a larger area, correspond to higher cheating gain. It is clear, that when d_{c-AP} is small (e.g., less than 4.5 m), the cheating gain is large ($\sim 11-13Mbps$). Slightly lower gains are observed if the cheating node resides further from the AP, but d_{c-l} is kept large ($\geq 10m$). Finally, for a larger d_{c-AP} in conjunction with a small d_{c-l}, the cheating gains are diminished. We would like to emphasize that, the above mapping is much more complicated than it seems. There are multiple factors that affect the cheating gain and the dependencies between these factors are complex. Distance is important but not *dominant*; for

3. Note that well behaved users may also exhibit similar behaviors in terms of the the qualities of their links to the AP; closer locations can result in higher RSSI values and thus, higher transmission rates can be sustained.

example, two nodes that are physically close to each other may have different communication channel due to environmental factors (e.g., obstacles). In this work, we focus on scenarios where cheating via manipulating CCA thresholds is feasible. Providing a definitive model that predicts the cheating gains is beyond the scope of this work.

In Figure 7 we present the temporal variations in throughput from a representative experiment. In particular, we use node 31 as an AP and nodes 22 and 14 as clients (Figure 1). We initiate fully saturated uplink traffic from both clients using *iperf* for 30 seconds. During the first ten seconds, both clients enjoy the same share of the throughput; this is a direct artifact of the fairness due to CSMA/CA. In the period between the 10^{th} and the 20^{th} seconds, node 14 misbehaves by increasing its CCA threshold from -80dBm to -50 dBm. We notice from Figure 7 that this results in a dramatic increase in the throughput of node 14. Meanwhile, node 22's throughput degrades significantly.

We observe that if the misbehavior is temporary, the effects are not long-lasting. As soon as the selfish user restores its default settings, the throughput of the rest of the clients quickly returns to the values under benign conditions. To understand this effect, recall that the selfish user follows the standard backoff mechanism with 802.11. After the settings are restored, within a short period of time, the greedy client enters the backoff state (senses energy on the medium). Then, the other users begin reducing their backoff counters; they gain access to the medium when their counter has reached the value of zero and at this point in time, fairness is restored[4].

Note also that even during the period where there is selfish behavior, the well-behaved nodes still obtain some throughput; this is directly attributed to the above reason i.e., packet losses can still occur for the misbehaving node and it can still enter the back-off phase.

Behavior with unsaturated UDP* selfish *traffic: Before examining the performance with TCP, we want to examine the impact of the varying volume of the cheating node's traffic load. In particular, we are interested into scenarios where the cheating node is not backlogged. When the uplink traffic demand of the selfish entity is low, one would expect that the effect (if any) on the other legitimate clients in the cell, is minimal. In order to showcase this, we perform experiments under different traffic conditions, but with the same topology. Figure 8 depicts our results for a representative experiment. Node 51 is configured as an AP with 3 associated clients (nodes 58, 32 and 34). Node 58 is the *selfish* user. We examine the following 3 scenarios. (i) All nodes use their default CCA settings. Nodes 32 and 34 have saturated uplink traffic, while node 58 has a traffic demand of $5Mbps$. (ii) Node 58 increases its CCA threshold to -50dBm, while the traffic patterns are the same as in (i). (iii) Node 58 keeps the increased CCA threshold and sends saturated traffic to the AP. As our results suggest, when the selfish user is not backlogged, increasing its CCA does not provide any throughput benefits as compared to that with default settings. The reason is that the cell has *enough* capacity to satisfy the user's demands. On the contrary, when node 58 is backlogged, more frequent medium access is required from its part to satisfy its uplink traffic demands. Consequently, the remaining users will suffer (nodes 32 and 34) as already seen. In general, with a higher traffic demand, the cheating node will access the channel with a higher frequency. Our experiments indicate that well-behaved clients suffer from unfairness and thus, receive a much reduced share of the throughput in such cases.

Behavior with TCP traffic: The use of TCP results in two somewhat conflicting effects from the perspective of a well-behaved user. On the one hand, since the selfish user accesses the medium more often (as discussed above), the TCP packets experience longer delays and round trip times (RTT); thus, the TCP congestion window does not increase as rapidly as one might expect under normal operations and the overall throughput suffers. On the other hand, the selfish client itself might experience loss of packets and this causes its TCP connection to reduce its congestion window. In other words, since with TCP the data rate is regulated by the congestion window, the selfish user will access the medium less often than it did in the UDP scenario when packet losses occur.

In order to quantify the impact of CCA tuning on TCP traffic, and in general the performance with different combinations of transport layer protocols, we conduct a large number of experiments. We use 90 different topologies using 15 different APs with 2 clients associated with each (a selfish client S and a legitimate client G) and consider all possible combinations of the two commonly used transport layer protocols, TCP and UDP. The misbehaving node employs its greedy strategy for the entire 30 second period (**abnormal** operation). The results are presented in terms of the average throughputs of the well behaved and the selfish nodes in Figure 9; 95% confidence intervals are also shown. We also show the performance during **normal** operations where both clients are using default settings. Note here that the measurements presented in Figure 9 encompass a large number of different topologies. However, all these topologies are *homogeneous*, in the sense of consisting of 2 clients with good quality links to the AP. Each client under benign settings gets a fair share of the medium (around 12Mbps each).

From the above results, it is evident that when the misbehaving client is sending UDP traffic its throughput gains are large. As one might expect, the impact is even higher when the well-behaved client is using TCP. The results show that significant gains are possible even if the link between the selfish user and the AP is lossy; this is because UDP does not reduce its sending rate upon experiencing packet losses. At the well behaved client, a lot of timeouts are triggered with TCP and the application throughput is extremely low (a few Kbps). When the misbehaving node uses TCP and the well behaved node uses UDP, the former is unable to achieve a significant gain in the throughput. This is a direct consequence of two factors (a) TCP regulates the sending rate thereby limiting the access opportunities for the selfish client and, (b) by increasing its CCA threshold, the selfish user can send more frequently, but when losses

4. The speed with which this process occurs depends on the quality of the link between the AP and the misbehaving client. If this link is lossy, the misbehaving client is likely to experience a packet loss quickly and enter the back-off state.

are experienced its TCP source backs off whereas the UDP source at the well-behaved user does not reduce its rate. When both the well-behaved node and the selfish node use TCP, the latter benefits. Both TCP sources back-off when there are losses; however, the selfish node is able to recover much faster since it is able to access the channel much more frequently. Figure 10 depicts the number of bytes sent per client for a representative configuration (AP-node 44, selfish client-node 13, well behaved-node 19).

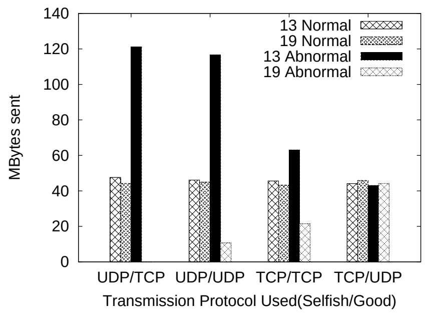


Fig. 10. Bytes sent with different transport layer protocols (AP:44, S:13, G:19).

To summarize, our experiments demonstrate that increasing the CCA threshold can lead to significant throughput benefits for the selfish client while hurting the other well-behaved clients, in a majority of the cases and with different transport layer protocols.

Network-wide experiments: It is interesting to examine what happens when all clients competing for the medium adopt the greedy strategy. In order to quantify the **network wide** degradation or improvement from such a strategy we perform a new set of experiments and we use the concept of the *Price of Anarchy (POA)* borrowed from game theory [22]. POA is a measure of how well/bad the players of a game do, when they play selfishly (e.g. increasing their CCA threshold), instead of according to a central authority (in our case using the default CCA threshold). POA can be defined using the following equation:

$$POA = \frac{\sum_{i=1}^{n} Greedy\ Throughput_i}{\sum_{i=1}^{n} Default\ Throughput_i} \tag{2}$$

where $Greedy\ Throughput_i$ is the throughput that client i enjoys when everyone is using the greedy strategy and $Default\ Throughput_i$ is the corresponding throughput of client i when everyone is using the default settings.

# Clients	POA
2	0.19
3	0.13
4	0.11

TABLE 1
POA decreases as we increase the number of clients.

We experimentally examine the network wide performance when all clients decide to act greedily. Table 1 presents the price of anarchy as computed from Equation 2 and our experimental results for the cases of 2, 3 and 4 clients. The degradation observed is significant as compared to the case with the default settings. In addition, we observe that the POE decreases - and the corresponding degradation increases - as we increase the number of clients affiliated with the AP. All clients ignore the signals of each other, resulting in many simultaneous transmissions; consequently there is a large number of collisions at the AP and the network performance is significantly degraded.

6 Detection system

In this section, we describe our scheme for detecting nodes that increase their CCA thresholds to gain an unfair throughput advantage in WLANs. We call our scheme CMD for Carrier sensing Misbehavior Detection system.

CMD is comprised of two sub-component modules: the first module, which we call TMM for Throughput Monitoring Module, aims to identify the set of *potential* cheating clients; note here that this set consists of those clients that are suspected of cheating but may not necessarily be real misbehaviors.

The second module LPM (for Low power Probing Module) tries to identify the real misbehaving clients. The key insight that motivates the design of LPM is that nodes that have increased their CCA thresholds may not be able to correctly decode low power probes. As we will see in what follows, *TMM significantly reduces the overhead that LPM can introduce.*

6.1 TMM: The Throughput Monitoring Module

As alluded to earlier, CMD sends probes in order to achieve its goal of detecting misbehaving users. Sending probes to all the clients associated with an AP can be prohibitive in terms of overhead. The goal of TMM is to identify the nodes that could be potentially cheating by increasing their CCA thresholds. Since the IEEE 802.11 is inherently fair, a node that gets a higher share of the available bandwidth could be a potential cheater. Note that it is not necessary that a node that gets a higher share of the bandwidth is essentially a cheater since different clients might have different traffic demands; the only conclusion that one can make is that such a possibility exists.

In order to identify the nodes that have a higher share of the medium, TMM monitors the volume of uplink traffic from each and every client. A node that is able to send a much larger volume of traffic is identified as a potential miscreant.

In order to demonstrate the effectiveness of this approach in terms of including misbehaving nodes in the set output by TMM, we perform the following experiment. We set up node 31 as an AP and include 3 associated clients (nodes 14, 22 and 37); each client sends saturated traffic to the AP. We measure the number of packets transmitted from each client to the AP for a period of 10 seconds under two different scenarios: (a) when no client cheats and, (b) when client 37 cheats. The results are presented in Table 2.

These results suggest that monitoring the traffic can be effective in identifying misbehaving nodes. However, recall

Client	14	22	37
Benign	9833	10521	10461
Cheating	320	521	21333

TABLE 2
TMM is effective with saturated traffic

Client	37	22	14
# Packets	1702	852	20322

TABLE 3
TMM can be mislead with unsaturated traffic

# Clients	3	5
Probing	26.1	21.8
No Probing	28.0	24.1

TABLE 4
Overhead with LPM

that in our experiments all clients have fully saturated uplink traffic. If the clients do not have saturated traffic they *may not* all have the same throughput under normal operations. In particular, if one of the clients produces a higher volume of uplink traffic, it will be mistakenly classified as a cheater if we were to just use TMM to identify the misbehaving nodes. To illustrate this we perform another experiment in which the same topology as in the previous case is used. The clients are now all benign. However, they have different application data rates: client 37 sends traffic at 2 Mbps, client 22 sends at 1 Mbps and client 14 at 24 Mbps.

Table 3 presents the results from this experiment. We observe that if TMM was used to classify nodes as cheaters, it would falsely conclude that client 14 is one. Thus, we need to further check if the nodes that are identified by TMM as potential cheaters are indeed cheaters or are legitimate recipients of higher throughputs; we do this using LPM (described later).

Implementation of TMM: We implement TMM in the user space. We develop a C application using `libpcap` [23]; the application is run at the AP and captures all the packets that arrive at its wireless interface. It internally maintains statistics in terms of how many packets are seen from each client in a Z second time window (we will refer to Z as the *monitoring window size*). It then compares the number of packets from each client in order to identify the potential cheaters; if the number of packets that a client transmits, exceeds its fair share by X percent (we will refer to X as the *deviation value*), it is considered to be a possible cheater. We defer a discussion on how to choose the values of X and Z to Section 8.

When the potential cheaters have been identified, TMM calls LPM (described in the next section) to determine whether or not a “potential cheater” is *indeed* a “cheater”. This implementation of TMM does not rely on an already available network monitoring system (for example, Ethereal or tcpdump). Instead, it computes the statistics online. In Algorithm 1, we give the high-level pseudocode of TMM.

```
Data: IP addresses of the AP's clients
Result: A potential cheater
begin
1    Every $Z$ seconds do:
2    for $i = 1$ to $num\_clients$ do
3        if $packets(i) > (1 + \frac{X}{100}) \cdot (\frac{total\_packets}{num\_clients})$ then
4            $Invoke\ LPM\ towards\ Client\ i$
         end
     end
end
```

Algorithm 1: Pseudocode for TMM

The periodic monitoring employed by TMM, *can* leave an open backdoor for short bursts of unfair access to the cheater. As an example, if the cheater has unsaturated, uplink traffic (which however appears in saturated bursts) it can employ the cheating strategy for these short bursts and potentially stay undetected. This is true, especially when Z is large. For small values of Z (e.g., a couple of seconds), such strategies cannot be successful, since the period over which the average throughput is computed is short. Nevertheless, in such cases of an undetected event, we are assured that the node does not get more than X% of its fair share of the medium for that period (line 3 of Algorithm 1). Thus, monitoring the average throughput enjoyed by the clients is seemingly sufficient as a *first level* of detection, if the monitoring window is kept fairly small. We will experimentally find the right values for Z in Section 8.

6.2 LPM: The Low Power Probing Module

The design of LPM is motivated by the observation that all the signals that arrive at the circuitry of a receiver with a received signal strength lower than the CCA threshold, are treated as noise; the receiver does not attempt to reconstruct packets from such signals [2]. Thus, a node that increases its CCA with the objective of increasing its throughput will not be able to correctly decode packets that are received with low powers. Thus, by having the AP probe the potential cheaters (determined by TMM) with low power packet transmissions, LPM achieves its goal of accurately identifying the real misbehaving clients.

A cheating node that increases its CCA towards obtaining a larger share of the available bandwidth, is likely to pick the *maximum* possible CCA without compromising on its connectivity with the AP[5]. The larger the CCA threshold, the higher are the number of possible ongoing transmissions that the carrier sensing logic ignores. If the CCA threshold is only increased slightly, the selfish node will not be able to achieve significant performance gains. Note here that due to this very reason, it is unlikely that nodes that are either distant from the AP (or have poor quality links) will be able to effectively launch the attack under consideration; they will not be able to increase their CCA thresholds significantly without compromising their connectivity to the AP.

Design of LPM: The new CCA threshold (chosen by a selfish node) is based on the RSSI from the AP under default operating conditions. If the AP transmits with lower powers (as compared with default settings), the RSSI value

5. We assume this to be the selfish behavior for now; other possible variants are discussed in section 9.

at a receiver is reduced. Going further, if this transmission power is considerably lowered, packets may arrive at the misbehaving node's antenna with an RSSI that is smaller than its *increased* CCA threshold. This is the key idea that drives LPM. The AP, using a reduced transmission power, sends a probe packet to each client that has been flagged as a potential misbehaving client by TMM. If a client node has increased its CCA to the extent that it exceeds the RSSI of the received probe packet, the client node cannot respond to the AP. The latter waits for a preset period of time for the client's response; if no response is received, the AP flags the client as a misbehaving node. To reduce the possibility of false alarms, LPM challenges the potential cheaters (listed by TMM) with successive *ICMP_ECHO_REQUEST* packets (64 bytes), sent using a reduced transmission power. The client is expected to reply to each probing packet that is received from the AP. If more than $W\%$ of the reply packets are missing from a particular client, the AP declares the client as a misbehaving client. In Section 7 we discuss how we choose W and the probing power such that there is a good trade-off between the false positive rate and accurate detection with our system.

```
Data: Client $i$ which has been flagged as a potential
      cheater by TMM
Result: Whether to declare it as a cheater
begin
1   $Ping(i, 10, Power_{probe})$
2   if more than W% of reply packets are missing then
3       $Declare\ Client\ i\ as\ a\ cheater$
    end
end
```

Algorithm 2: Pseudocode for LPM

TMM reduces the probing overhead due to LPM. We point out that LPM increases the overhead by sending probe packets on the medium. If the AP were to probe all the clients, then the performance degradation could be significant, especially when the number of clients is large. Table 4 shows the degradation in the aggregate throughput of an AP when (i) all the clients had fully saturated uplink traffic and (ii) the AP was constantly probing the clients in a round robin fashion with 10 probe packets sent to each client during a probe cycle.

We observe that if there are 3 clients associated with the AP the degradation is about 7 %; when there are 5 clients, the degradation is about 9.5%. As the number of clients increases, the degradation is higher; therefore, it is crucial to reduce the number of clients that LPM checks for real cheaters. Based on this, it is clear that TMM plays an important role in our system.

Note also that currently, we use the 64 byte *ICMP_ECHO_REQUEST* messages as probes; it is possible to reduce the overhead by creating special probe messages that are of smaller size. However, this will increase the complexity of the implementation (the current implementation is described below) and may require modifications to the 802.11 driver/firmware.

Implementation details of LPM: We have implemented LPM in the user space, on top of the wireless NIC's driver. It is run at the access point. Our implementation uses a shell script that invokes the *ping* application [24] to probe the clients. More specifically, the script consists of a loop which parses the list of clients that are flagged as potential cheaters by TMM. We set the transmission power of the "ping" packets using the *iwconfig* command. Based on the results of the ping trials, LPM decides on whether a client is a cheater. This implementation is generic in that it can be run in conjunction with most commodity wireless NIC drivers.

For our Atheros cards, which use the MadWifi driver, we have also implemented our own probing utility using the Click Modular Router [25]. We use the `ICMPPingSource` and `ICMPPingResponder` elements to implement a *probe sender* and a *probe receiver*, respectively. The `SetTXPower` element enables us to set the transmission power for each $ICMP$ packet sent out by LPM. This element simply sets the Wifi TXPower Annotation flag on the packet to be sent, and we do not need to subsequently call *iwconfig* to set the power.

7 An Analytical Model to Derive System Parameters

The design of LPM is based on the observation that a cheating node with an increased CCA is unlikely to respond to probe packets sent by the AP with a low transmission power. There are two cases, however, where LPM may not lead to correct diagnosis: **(i)** Benign clients located at the border of the AP's coverage area may not be able to respond to low power probe packets sent from the AP; these packets are likely to arrive at their circuitry with an RSSI lower than the default threshold CCA_{def}. This results in what we call *false positives*. (Note here that even though the links to such clients are likely to be poor, some of these clients may be getting a higher share of throughput in unsaturated traffic conditions). **(ii)** Misbehaving nodes could be so close to the AP that in spite of the AP using reduced transmission powers, probe packets can still reach their circuitry with an RSSI higher than their increased CCA value. In this case, the misbehaving node is not identified i.e., we have a *false negative*. In this section we analyze the performance of our system to determine various parametric inputs to CMD such that the false positive and false negative rates are kept low.

Propagation Model: In order to analytically determine the false positive and the false negative rates, we need to assume a propagation model. We calculate the received power P_r at distance r with transmission power P to be:

$$P_r = \frac{P}{r^{\alpha}} \cdot Y, \tag{3}$$

where α is the path loss exponent and Y is a random variable that is log-normally distributed. The random variable Y models the shadow fading effects and it has a mean value of one and a standard deviation equal to the shadow fading variation (obtained from measurements). The above model has been shown to be fairly accurate in indoor settings [26] [27].

False positives: We first compute the false-positive rate.

The probability $f(P,r)$ that a probe packet from the AP arrives at distance r with an RSSI below CCA_{def} is given

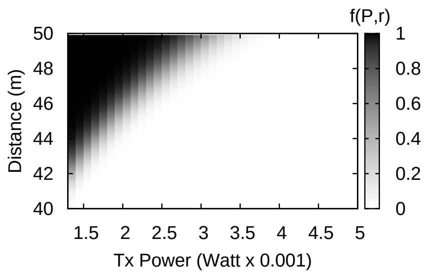


Fig. 11. $f(P,r) = Pr\{signal(P,r) < CCA_{def}\}$.

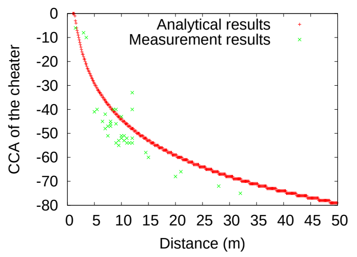


Fig. 12. The theoretical and practical CCA_{cheat}.

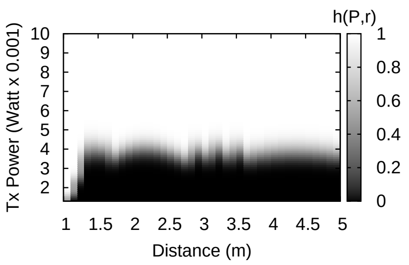


Fig. 13. $h(P,r) = Pr\{signal(P,r) > CCA_{ch}\}$.

by:

$$f(P,r) = Pr\{\frac{P}{r^{\alpha}} \cdot Y < CCA_{def}\} = Pr\{Y < \frac{CCA_{def}}{P} \cdot r^{\alpha}\}$$
$$= \frac{1}{2} + \frac{1}{2} \cdot erf(\frac{ln(\frac{CCA_{def} \cdot r^{\alpha}}{P}) - \mu}{\sigma \cdot \sqrt{2}}), \quad (4)$$

where μ and σ are the parameters of the log-normal distribution (computed from the mean and the measured standard deviation). We plot this probability in Figure 11. In generating this probability, the following values are used to derive the results: (i) CCA_{def}=-80dBm, (ii) the shadow fading variation is 5dBm (as measured from our testbed), and (iii) $\alpha = 5$, which is a typical value for the path loss exponent for an indoor environment [26] [28]. The figure shows that with extreme low power operations (1.5 mW), the probability of violating the default CCA threshold is extremely high (false positive); with moderately low powers (3 mW), this same probability is almost zero up to distances of 50 meters.

Equation (4) gives the probability that a packet arrives at the client's circuitry, after traveling distance r, with power less than CCA_{def}. Let us assume that LPM transmits 10 probe packets and expects n replies. Let $pr^{pos}(P,r,n)$ denote the probability that fewer than n probe packets[6] arrive at a distance r with an RSSI greater than CCA_{def}. This probability is given by:

$$pr^{pos}(P,r,n) = \sum_{k=1}^{n}(1-f(P,r))^{k-1} \cdot f(P,r)^{10-k+1} \quad (5)$$

In order to calculate the false detection rate at distance r when the transmission power is P, we need the spatial distribution of nodes $s(r)$. As discussed in Section 5, nodes tend to stay close to the AP in reality. In order to get numerical results, a possible spatial distribution that can be used based on the previous observation is $s(r) = \frac{1}{ln(50) \cdot r}$, for $1 \leq r \leq 50\ m$ and zero otherwise[7] (the constant $\frac{1}{ln(50)}$ is chosen to assure that function s is a valid probability density function). With this spatial distribution model, the false positive rate $\pi^{pos}(P,r,n)$ at a distance r, when the transmission power is P is given by:

$$\pi^{pos}(P,r,n) = pr^{pos}(P,r,n) \cdot s(r) \cdot \Delta r|_{\Delta r \to 0} \quad (6)$$

6. We assume that the channel is reciprocal and thus, if the probe message is correctly received, the corresponding $ICMP\_ECHO\_REPLY$ packet will be received with very high probability; this assumption ensures the tractability of our analysis.

7. Nodes are expected to have a minimum distance -e.g. 1m - from the AP which in commercial hotspots are deployed mainly on ceilings. Note that our analysis can incorporate any other spatial distribution.

We can then compute the overall false positive rate $\pi_p(P,n)$ when the AP is using transmission power P and when LPM expects n replies to its probes by integrating over the area of the cell:

$$\pi_p(P,n) = \int_0^{\infty} pr^{pos}(P,r,n) \cdot s(r)\, dr \quad (7)$$

False negatives: Similar steps as above are taken in order to compute the false negative rate. However, we first need to estimate the CCA threshold, that a cheating node at distance r is likely to use. The goal of the selfish client is to avoid as many transmissions as possible by increasing its CCA while maintaining its connectivity with the AP (note that this is when the AP is using the default power P_{def}, i.e., under default operations). The CCA chosen according to this strategy can be computed by solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & CCA_{cheat}(r) & (8)\\ \text{subject to} \quad & Pr\{\frac{P_{def}}{r^{\alpha}} \cdot Y > CCA_{cheat}(r)\} = 1 & (9)\\ & CCA_{cheat}(r) \in \{-80, -79, ..., 1, 0\}dBm & (10) \end{aligned}$$

Solving the above optimization problem for various distances r, we obtain the results shown in Figure 12. We present in the same figure, the corresponding $CCA_{cheat}(r)$ (the CCA threshold tuned as per the same strategy) measured from our testbed; for a given location of the cheater we increase the CCA threshold to the extent possible without compromising the connectivity with the AP. The results indicate that the analytical results match reasonably well with the measurement results; the coefficient of determination R^2 [29] is calculated to be equal to 0.71.

Having computed $CCA_{cheat}(r)$, we now proceed to calculate the false negative rate. We first calculate the probability $h(P,r)$ that a signal transmitted from the AP with power P arrives at distance r with a RSSI greater than $CCA_{cheat}(r)$:

$$h(P,r) = Pr\{\frac{P}{r^{\alpha}} \cdot Y > CCA_{ch}(r)\} = Pr\{Y > \frac{CCA_{ch}(r)}{P} \cdot r^{\alpha}\}$$
$$= \frac{1}{2} - \frac{1}{2} \cdot erf(\frac{ln(\frac{CCA_{ch}(r) \cdot r^{\alpha}}{P}) - \mu}{\sigma \cdot \sqrt{2}}) \quad (11)$$

In Figure 13 we plot $h(P,r)$ for various AP transmission powers and distances from the AP (using the same parameters as previously) and $CCA_{cheat}(r)$ computed as the solution to the optimization problem defined in (8)-(10). We observe that if the cheater is extremely close to the AP (≈ 1 m), there is no way of detecting it with low power probes.

However, if the cheater is further than 1.5 meters, the use of a transmission power that is lower than say 3.5 mW can lead to an extremely high probability of detection, i.e., the probability that the signal is higher than the CCA set by the cheater is almost zero.

Given $h(P,r)$, we now calculate the probability $pr^{neg}(P,r,n)$ that no fewer than n packets arrive at distance r with an RSSI greater than $CCA_{cheat}(r)$:

$$pr^{neg}(P,r,n) = \sum_{k=n}^{10} h(P,r)^k \cdot (1-h(P,r))^{10-k} \tag{12}$$

Using the spatial distribution of the nodes $s(r)$, we can calculate $\pi^{neg}(P,r,n)$, the false negative rate at distance r when the transmission power of the AP is P to be:

$$\pi^{neg}(P,r,n) = pr^{neg}(P,r,n) \cdot s(r) \cdot \Delta r|_{\Delta r \to 0} \tag{13}$$

Integrating over the whole area, we get the overall false negative rate $\pi_n(P,n)$ when the AP transmits with power P and LPM expects n responses to its probes:

$$\pi_n(P,n) = \int_0^\infty pr^{neg}(P,r,n) \cdot s(r)\, dr \tag{14}$$

Equations (7) and (14) provide the false positive and false negative rates of our system. These results also provide insights on the appropriate values for $Power_{probe}$ and n; these values should be chosen so as to satisfy a specific performance criterion. In short, we seek to minimize these probabilities; however, it is unlikely that they are both minimized together. Hence, we minimize the sum $\pi_p(P,n)+\pi_n(P,n)$. Solving this minimization problem yields $n = 9$ and $Power_{probe} = 3.3mW$. This means that in the LPM engine we need to set $W = 10\%$ (since 10 probes were sent) and $Power_{probe} = 3.3mW$. In Figure (14) we present the ROC curve (Receiver Operating Characteristics) for the case $n = 9$ and we point out the operating point which corresponds to $Power_{probe} = 3.3mW$. Each point on this curve corresponds to a different $Power_{probe}$. Increasing $Power_{probe}$ increases false negatives; decreasing it will increase false positives. The operating point is the one that minimizes the aforementioned objective function. The corresponding false positive rate and false negative rates are: $\pi_p = 0.0053$ and $\pi_n = 0.054$. Note that with these settings, our detection system is able to achieve high detection accuracy.

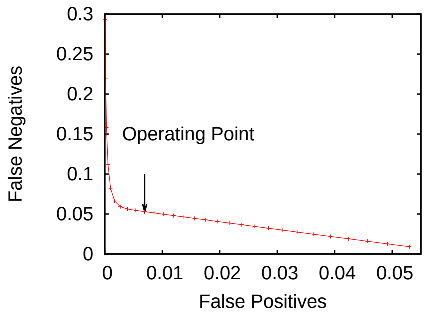


Fig. 14. The analytical ROC curve of our system.

8 Evaluation of CMD

In this section, we evaluate CMD.

Evaluation of the TMM module: First, we perform experiments to evaluate how TMM performs with various combinations of its input parameters; in particular, we consider (a) the monitoring window size Z and, (b) the deviation value $X\%$. Ideally, we want TMM to (i) flag all cheating nodes as potential cheaters *and* (ii) minimize the number of well-behaved nodes that are included in the set of potential cheaters. To evaluate the performance of TMM, we conduct the following two sets of experiments.

(a) Monitoring legitimate traffic: In this set of experiments we monitor the traffic at the AP when no clients cheat and all of them have fully saturated uplink traffic. We vary both the monitoring window size Z and the deviation $X\%$. The *false alert rate*, which represents the probability that a well-behaved client is flagged as a potential cheater, is depicted in Figures 15-17; in these experiments, the numbers of clients associated with the AP are 2, 3 and 4, respectively.

From the results, we observe that when the deviation is chosen to be smaller than 20% the false alert rate can be very high, especially when the monitoring window size is small. For instance, when the deviation is set to 10% and the monitoring window size is 1 second, a well-behaved client is mistakenly flagged as a potential cheater with a likelihood higher than 30%. However, if we increase the deviation, the false alert rate decreases. When the deviation is set to 30% or higher, the false alert rate is very small. The results are somewhat expected since small deviations in the expected fair share of throughput are likely; furthermore, transients are possible if the monitoring window size is not sufficiently large. Reducing the false alert rate will reduce the overhead incurred due to probing with LPM.

(b) Monitoring the cheating nodes' traffic: In this set of experiments we monitor the traffic at the AP under the presence of cheating nodes. Again, all clients have fully saturated uplink traffic. In this case, we are interested in the false negative rates of TMM; in other words, we seek to measure the probability that TMM does not include a *real* cheater in its output set. Figures 18-20 depict the probabilities that a cheating node is successfully identified as a potential cheater. From the results, we observe that when we use relatively small deviations (smaller than 30%) the TMM module almost always flags the cheating node as a potential cheater. If however, a deviation value higher than 30% is used, the false negative rate increases.

The experimental results with both scenarios suggest that there is a tradeoff between the detection accuracy and the deviation value. Small deviation values help identify the cheating nodes but they may lead to high false alert rates under benign conditions; on the other hand, large deviation values help reduce the false alert rate but may result in missing some cheating nodes. In the current version of TMM, we set the monitoring window size to be 1 second and the deviation value to be 30%. Based on the experimental results, these values achieve a good balance between the false alert rate and the false negative rate of TMM.

Evaluation of the LPM module: LPM determines whether a potential cheater reported by TMM is indeed a

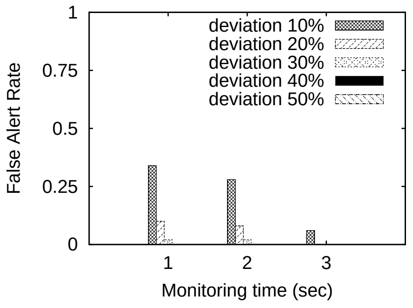


Fig. 15. TMM false alert rate when there is no selfish user (2 clients).

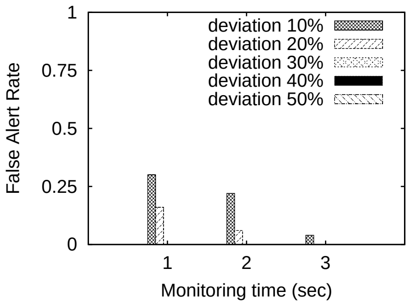


Fig. 16. TMM false alert rate when there is no selfish user (3 clients).

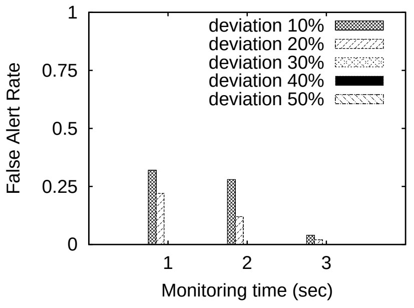


Fig. 17. TMM false alert rate when there is no selfish user (4 clients).

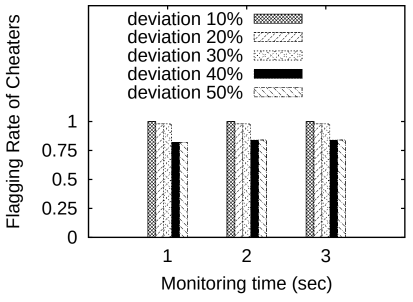


Fig. 18. TMM flagging rate of cheaters when there is a greedy client (2 clients).

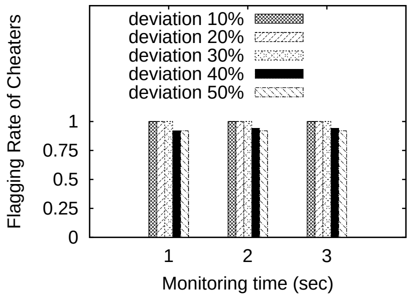


Fig. 19. TMM flagging rate of cheaters when there is a greedy client (3 clients).

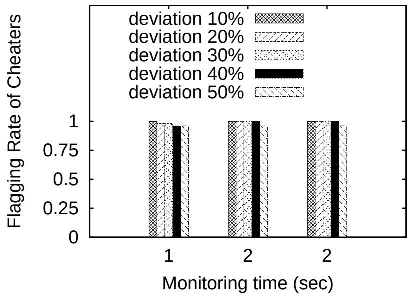


Fig. 20. TMM flagging rate of cheaters when there is a greedy client (4 clients).

cheater. We perform another set of experiments to quantify its detection accuracy. We experiment with a variety of configurations that take into account both saturated and unsaturated uplink traffic. In particular we experimented with 132 configuration tuples. We utilize `iperf` to generate uplink traffic. The cheating node always has saturated traffic and misbehaves shortly after the initiation of the experiment (8-10 seconds approximately). Each experiment lasts for 1 minute. We vary the transmission power of the probe packets between 3, 4, and 5 dBm. Recall that our analysis in Section 7 suggests a probe power of $3.3mW$; this corresponds to approximately $5dBm$. We compute the false positive and false negative rates with the LPM module. Note that since LPM takes the output of TMM as its input, these rates are the false detection rates for the whole system (the output of LPM is the output of CMD)[8]. The results are presented in Table 5.

$Power_{probe}$	False positive rate	False negative rate
5dBm	0.015	0.060
4dBm	0.015	0.030
3dBm	0.045	0.015

TABLE 5
Detection accuracy of CMD

From Table 5, we note that our system produces low false positive rates and low false negative rates in real experiments; even when the transmission power of the probe packets is varied, the maximum false positive and the maximum false negative rates are no higher than 4.5% and 6%, respectively. We also observe the tradeoff between false positive rates and false negative rates as we reduce (or increase) the probing power; if we keep reducing the probing power, the false positive rate increases while the false negative rate decreases. From among the three probing powers used, the sum of false positive rate and the false negative rate is minimized when $Power_{probe}$ is $4dBm$. This value is slightly lower than the one derived with the analysis in Section 7. The reason for this is that the assumed propagation model and its parameters (i.e., path loss exponent) or the spatial distribution of nodes $s(r)$ with the analysis, may not fit with the characteristics of our testbed with very high fidelity. Furthermore, in our analysis we focus on the performance of LPM, without considering the impact of TMM. It is hard, if not impossible, to model the interactions between the two modules accurately. This would require $s(r) \cdot \Delta r$ to capture the probability of TMM reporting a node, at distance r, as a potential cheater; this is difficult because it requires the knowledge of the traffic patterns of all clients (whether they send saturated traffic or not and their application data rates) at each location. *In spite of these limitations, note that the false positive rate and the false negative rate analytically derived (i.e., π_p and π_n in Section 7) are very close to what is observed with experimental results on the real testbed.*

In our experiments LPM mistakenly declares a few well-behaved nodes as cheaters; this happens especially when some of the clients have unsaturated uplink traffic. As discussed in Section 5, clients far away from the AP cannot gain much by applying the considered selfish strategy because they cannot increase their CCA thresholds to a significant extent. In the presence of unsaturated traffic, some well-behaved clients that are far away from the AP

8. We refer the reader to the Appendix for a derivation of the false positive/negative probabilities of CMD.

are wrongly flagged as potential cheaters by TMM if their application data rates are higher than that of those that are closer to the AP. Consequently with LPM, the probe packets from the AP may reach these clients with a RSSI below $CCA_{def} = -80dBm$. Thus, these well-behaved clients are unable to recognize these packets and send responses to the AP. However, our experiments demonstrate that such occurrencies are rare given that the poor quality of the links to such clients limits the throughput that they can achieve.

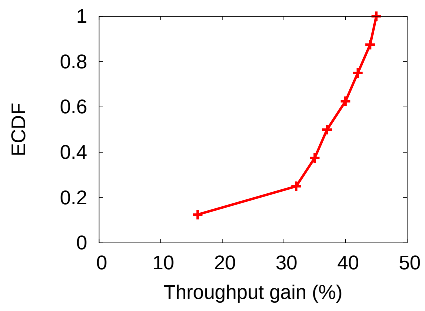


Fig. 21. A cheating node that escapes detection can gain up to 47% of its faire share.

We observe that the false negative rate is about 6% when the transmission power of probe packets is $5dBm$. As we reduce this power, the false negative rate decreases significantly. For instance, when probe packets are transmitted with power $3dBm$, the false negative rate drops to about 1.5%. Interestingly, if we further reduce the transmission power of probe packets to $1dBm$, all cheating nodes flagged by TMM are successfully reported as cheaters[9].

Selfish gain under false negative detection: Since there are cases where false negatives appear, we are interested in finding the actual gain of a cheating node that avoids detection. This essentially reflects the effect of false negatives in the network. Recall that currently CMD is configured with $X = 30\%$ and $Z = 1sec$. Figure 21 presents the empirical CDF of the selfish gain (percentile deviation from the fair share) in the case of false negatives. The same 132 configuration tuples as above were used with the three different $Power_{probe}$ levels (in 396 trials). We observe that the cheating gains range between 16% and 47%. Scenarios with gains smaller than 30% correspond to cases where LPM was not triggered by TMM. Once TMM flags a cheating node, LPM is responsible for detecting actual selfish users, and as alluded to above, reducing the probing power increases the efficiency of our system with regards to the false negatives. Note here that, even though a cheating gain of 47% appears to be fairly high, the occurrencies of this event are rare. In particular, there were in total 14 scenarios (out of 396) where false negatives were observed. TMM was responsible for 3 of them, while LPM was responsible for the other 11 occurrences[10]. We defer a discussion on the effects of the false positives on the total network throughput to the next section, after describing possible mitigation schemes.

9. However we expect that such a low $Power_{probe}$ can lead to a high false positive rate.

10. TMM contributes to the first term of the right hand side of Equation 2 at Appendix, while LPM contributes to the second term.

9 Miscellaneous Issues

Mitigating the effect of selfish tuning of the CCA: The goal of our work is to *detect* users that selfishly increase their CCA thresholds in order to obtain throughput gain. *Mitigating* the effects of such misbehaving nodes is not the focus of our study; however, we deliberate on possible ways of overcoming the adverse effects of such cheaters. The most aggressive, yet the simplest, solution is to punish a cheating client by disassociating it completely from the AP. There are other mitigation approaches that are less harsh. As an example, the AP can choose to reduce its transmission power, which forces the cheating client to decrease its CCA threshold if it wants to further communicate with the AP. Alternatively the AP may intentionally drive down the throughput of such misbehaving clients. In particular, the AP could choose to "not send" MAC layer ACKs to the cheating node for some of its frames. As a result, the cheating node has to back off with a larger contention window; this in turn, increases the opportunity of access to the well-behaved nodes. Implementation of this approach is challenging because currently most commodity NICs implement MAC layer acknowledgments in the firmware.

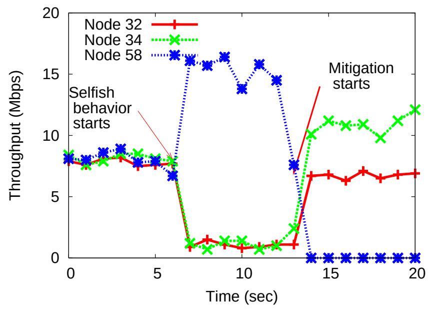


Fig. 22. Reducing the transmission power can restore throughput in well behaved clients.

Clearly, all of the above proposed approaches can alleviate the effect of a cheating node. It is up to the network administrator to choose the appropriate strategy. Note that each possible strategy has its pros and cons. For example, at one extreme is the approach of disassociating the selfish user upon detection. A much milder approach is the filtering of the MAC layer ACKs. In this case, the misbehaving node still enjoys some throughput and more importantly is not *forced* to decrease its CCA threshold. Between the two above extreme schemes is the transmission power control scheme. With this approach the selfish user will be forced to decrease its CCA threshold in order to stay connected with the AP.

We perform experiments by tuning the transmission power of the AP. In the representative time trace presented in Figure 22, node 51 is set as an AP, while backlogged nodes 58 (selfish), 32 and 34 are associated with it. For the first 6 seconds of our experiment every client uses the default CCA thresholds, and the AP uses the maximum transmission power (18dBm). After the 6^{th} second, and for the next 6 seconds, node 58 increases its CCA threshold; during this time AP 51 still uses the maximum transmission power. At $t = 13sec$, the AP, 51 reduces the transmission power to 1dBm. As our results show, reducing the transmission power used by the AP causes the greedy client to disassociate from the cell. If the greedy user increases its CCA value it will be able to re-associate with the AP and enjoy its fair share. This scheme can be thought of as a passive scheme because of this.

Note here that, reducing the transmission power of the AP, can have an impact on the well behaved clients as compared with the performance under benign settings. This is due to the fact that the link "AP to client", might experience a degradation due to the lower transmission power. If we take a closer look at the time trace for node 32, we see that despite the fact that it enjoys an improved performance with regards to the "selfish behavior" period, it does not obtain the throughput that is enjoyed in benign settings. This effect can be overcome by performing power control in a finer granularity (e.g., per client).

The effects of false positive detection: When a selfish node is detected, the mitigation scheme will be triggered. The same is true in the case of a false positive. The difference is that in the latter case the mitigation system will take action against a well behaved node. Depending on the semantics of the scheme deployed, the effects of a false positive detection will be different.

With the aforementioned transmission power control scheme, the throughput of the falsely classified node will be disassociated. However, the other nodes will not be affected and they will enjoy at least the same throughput prior to the activation of the mitigation scheme. In fact when there is saturated traffic demand, the throughput that each of them enjoys will increase. This is because they will fairly *share* the *spare* capacity available. Note here that, the disassociated node can possibly re-associate with the AP by decreasing its CCA. However, this may not be possible for some of the clients at the edge of the cell. Recall however, that given the extremely low false positives rate, the chances of this are extremely low.

Other cheating strategies: Throughout our study, we have assumed that a cheating node always chooses the maximum CCA threshold that guarantees its connectivity with the AP. This assumption is reasonable only if the cheating node is extremely greedy (the strategy enables the node to ignore as many transmissions as possible). If the misbehaving node knows that CMD has been deployed, it might set a CCA threshold lower than that to evade detection. A less significant increase in CCA will however have a lower impact on the network, as seen in Figure 23. In this figure, we plot the selfish gains for 2 different representative scenarios and for different CCA thresholds employed by the misbehaving node. Scenario A refers to node 51 being set as an AP and nodes 32 and 34 being the associated clients (node 32 is the cheating node). Scenario B refers to node 31 being the AP and nodes 37 and 22 being its clients (node 22 is the selfish user). For scenario A, any value different from CCA_{cheat} ($\sim -45dBm$), will result in significantly lower gain[11]. When we set $CCA < CCA_{cheat}$, the selfish user cannot ignore ongoing transmissions to the extent desired; if we use $CCA > CCA_{cheat}$, the link quality to the AP is degraded. For extremely high CCA values (e.g., -35dBm) the connectivity is completely compromised and thus, the throughput is nullified (translating to a negative gain). Thus, there is an inherent trade-off between the performance gain and the possibility of detection that the cheater has to consider.

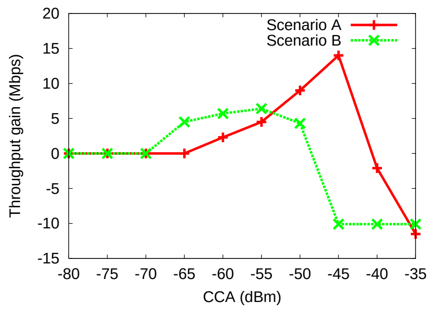


Fig. 23. Less aggressive cheating strategies result in lower gains.

For scenario B, the cheating node can use slightly lower values for CCA as compared with CCA_{cheat}. The range of these values is $\sim 15dBm$. In such cases, the selfish user might keep its CCA lower than the CCA_{cheat}, in order to decrease the probability of being detected. Note however that, it is still possible to detect the misbehavior by further reducing the transmission power of the probe packets. Of course, this may lead to higher false positive rates. However, from Figure 14 we notice that even if we use the lowest transmission power considered, the false positive rate is still very low (relative to the specific spatial distribution).

From the results from Figures 6 and 23, it is clear that a priori estimation of the CCA value that will provide the *optimal* tradeoff for the cheater is extremely complicated. **Fine tuning the CCA threshold to achieve a specific selfish gain and/or guarantee detection avoidance is hard due to the wireless propagation effects and the dependency on the topological properties.**

Finally, we would like to point out that, a *one-size-fits-all* solution is hard to design. Our system, cannot deal with attacks that are more sophisticated than **direct** CCA manipulations. For instance, a user who utilizes two different CCA thresholds, one for data transmissions and one for probe receptions cannot be detected. However, note that such intelligent strategies are not easy to implement on commodity cards. Rapid switches between thresholds introduce large overheads. Moreover, it is difficult for the cheater to know when probes are being sent by an AP.

10 Conclusions

In this paper we identify a new, powerful selfish behavior in 802.11 networks: a misbehaving node increases its CCA to improve its chances of accessing the medium. CCA tuning has been considered previously towards providing network wide performance enhancements; this is the first study that considers the misuse of this capability. With extensive experimentation on a real testbed, we show that the selfish behaviors considered can cause extremely unfair allocations of the wireless medium. We develop a detection scheme

11. Since our measurements are taken using steps of 5dBm, the actual value for CCA is within 5dBm from the one measured.

for WLANs that we call CMD for Carrier sensing Misbehavior Detection. We mathematically analyze its detection accuracy. We also implement CMD on an indoor wireless testbed. Through experiments we demonstrate that CMD detects such selfish clients in WLANs with extremely high accuracy and with low false positive rates.

Acknowledgments

The authors would like to thank Dr. Konstantina Papagiannaki from Intel Research for providing the source code of the prototype driver.

References

[1] M.Heusse, F.Rousseau, G.Berger-Sabbatel, and A.Duda. Performance anomaly of 802.11b. In *INFOCOM*, 2003.
[2] V. Mhatre, K. Papagiannaki, and F. Baccelli. Interference Mitigation through Power Control in High Density 802.11 WLANs. In *IEEE INFOCOM*, 2007.
[3] I. Broustis, K. Papagiannaki, S. V. Krishnamurthy, M. Faloutsos, and V. Mhatre. MDG: Measurement-Driven Guidelines for 802.11 WLAN Design. In *ACM MOBICOM*, 2007.
[4] P. Kyasanur and N. Vaidya. Detection and Handling of MAC layer misbehavior in wireless networks. In *DSN*, 2003.
[5] S. Radosavac, J. S. Baras, and I. Koutsopoulos. A framework for MAC protocol misbehavior detection in wireless networks. In *WiSe*, 2005.
[6] M.Raya, J-P.Hubaux, and I.Aad. DOMINO: A System to Detect Greedy Behavior in IEEE 802.11 Hotspot. In *MobiSys*, 2004.
[7] M. Cagalj, S. Ganeriwal, I. Aad, and J.-P. Hubaux. On selfish behavior in CSMA/CA networks. In *INFOCOM*, 2005.
[8] O.Queseth. The effect of selfish behavior in mobile networks using CSMA/CA. In *VTC*, 2005.
[9] J. Konorski. Multiple access in ad hoc wireless LANs with noncooperative stations. In *NETWORKING*, 2002.
[10] ANSI/IEEE 802.11-Standard. 1999 edition.
[11] Ath5k project. http://madwifi.org/wiki/About/ath5k.
[12] GNU radio trac. http://gnuradio.org/trac.
[13] USRP SDR platform. http://www.ettus.com.
[14] J.Lee, S.Choi, and H.Jung. Analysis of User Behavior and Traffic Pattern in a Large-Scale 802.11a/b Network . In *WiNMee*, 2005.
[15] B. O'hara and A. Petrick. *IEEE 802.11 Handbook, a Designer's Companion*. IEEE Press, Second Edition, ISBN 0-73-814449-5.
[16] U. Paul, S. R. Das, and R. Maheshwari. Detecting Selfish Carrier-Sense Behavior in WiFi Networks by Passive Monitoring. In *IEEE/IFIP DSN*, 2010.
[17] Soekris-net4826. http://www.soekris.com/net4826.htm.
[18] The MAdWiFi driver. http://madwifi.org.
[19] UC Riverside Wireless Testbed. http://networks.cs.ucr.edu/testbed/.
[20] K.Jamieson, B.Hull, A.Miu, and H.Balakrishnan. Understanding the Real-World Performance of Carrier Sense. In *ACM SIGCOMM Workshops*, 2005.
[21] J. Zhu, B. Metzler, X. Guo, and Y. Liu. Adaptive CSMA for Scalable Network Capacity in High-Density WLAN: A Hardware Prototyping Approach. In *IEEE INFOCOM*, April 2006.
[22] E. Koutsoupias and C. Papadimitriou. Worst-Case Equilibria . In *16th Annual Symposium on Theoretical Aspects of Computer Science*, 1999.
[23] PCAP Unix man page. http://www.tcpdump.org/pcap3_man.html.
[24] Ping Linux Man Page. http://linux.die.net/man/8/ping.
[25] Click Modular Router. http://read.cs.ucla.edu/click/.
[26] S. Zvanovec, P. Pechac, and M. Klepal. Wireless LAN Networks Design: Site Syrvey or Propagation Models? In *Radioengineering, Vol. 12, No. 4*, Dec. 2003.
[27] T. S. Rappaport. *Wireless communications principles and practices,*. Prentice Hall, 2002.
[28] Path-loss. http://en.wikipedia.org/wiki/Path_loss.
[29] N.R. Draper and H. Smith. *Applied Regression Analysis*. Wiley-Interscience. ISBN 0-471-17082-8.

Appendix

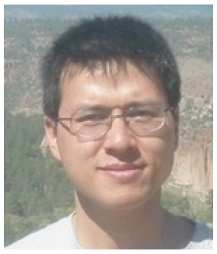
CMD consists of two concatenated sub-systems, TMM and LPM. In order for CMD to result in a false positive, two *conditions* need to be satisfied. First, TMM must flag a well behaved user as a potential cheater (i.e., a false positive must occur with TMM). Second LPM must result in a false positive as well. Since the above events are independent, the total false positive proabitlity of CMD, $\pi_p(CMD)$ is given by:

$$\pi_p(CMD) = \pi_p(TMM) \cdot \pi_p(LPM) \quad (15)$$

In order to express the total false negative probabitlity of CMD, $\pi_n(CMD)$, we need to calculate the probabilities of the following two events: (i) TMM fails to identify a cheater and (ii) TMM *flags* the selfish user as a potential cheater but LPM fails to detect it (i.e., false negative occurs with LPM). Event (ii) is composed of two independent events and thus, $P(event(ii)) = (1-\pi_n(TMM))\cdot\pi_n(LPM)$. Since, events (i) and (ii) are mutually exclusive we have:

$$\pi_n(CMD) = \pi_n(TMM) + (1 - \pi_n(TMM)) \cdot \pi_n(LPM) \quad (16)$$

Konstantinos Pelechrinis Konstantinos Pelechrinis received his PhD from the Computer Science department of University of California, Riverside, in 2010. Previously he obtained his MSc degree from the Computer Science department of University of California, Riverside in 2008 and the diploma of Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 2006. He is an Assistant Professor at the SIS faculty of the University of Pittsburgh since Fall 2010. He has also held research positions at LANL, Thomson Research Labs Paris and MSR Cambridge. He was a visiting researcher at the University of Thessaly during Fall 2008. His research interests include wireless networking, especially security - related issues that span the full protocol stack. He is involved in protocol design, real world experimentation and performance analysis. He is also interested in mathematical foundations of communication networks.



Guanhua Yan Guanhua Yan obtained his Ph.D. degree in Computer Science from Dartmouth College, USA, in 2005. From 2003 to 2005, he was a visiting graduate student at the Coordinated Science Laboratory in the University of Illinois at Urbana-Champaign. He is now working as a Technical Staff Member in the Information Sciences Group (CCS-3) at the Los Alamos National Laboratory. His research interests are cyber-security, networking, and large-scale modeling and simulation techniques. He has contributed about 30 articles in these fields.

Stephan Eidenbenz Stephan Eidenbenz received his Ph.D. degree in Computer Science from the Swiss Federal Institute of Technology (ETH) in Zurich in 2000. He is now a team leader in the Information Sciences Group (CCS-3) at the Los Alamos National Laboratory, where he leads the Multi-scale Integrated Information and Telecommunications System (MIITS) project that models and simulates large-scale com- munication networks. His research interests are in wire-line and wireless networking, sensor networks, selfish networking, infra-structure modeling, discrete event simulation, computational geometry, and algorithms. He has published about 50 articles in these fields.

Srikanth V. Krishnamurthy Srikanth V. Krishnamurthy received his Ph.D degree in electrical and computer engineering from the University of California at San Diego in 1997. From 1998 to 2000, he was a Research Staff Scientist at the Information Sciences Laboratory, HRL Laboratories, LLC, Malibu, CA. Currently, he is a Professor of Computer Science at University of California, Riverside. His research interests are primarily in wireless networks, network security and Internet technologies. Dr. Krishnamurthy is the recipient of the NSF CAREER Award from ANI in 2003. He has also co-edited the book “Ad Hoc Networks: Technologies and Protocols” published by Springer Verlag in 2005. He served as the editor-in-chief for ACM MC2R between 2007 and 2009 and is a senior member of the IEEE.