Detection and Punishment of Malicious Wireless Stations in IEEE 802.11e EDCA Network

Yong woon Ahn*, Albert Mo Kim Cheng*
Department of Computer Science
University of Houston
Houston, TX 77204
{yahn, cheng}@cs.uh.edu

Jinsuk Baek, Paul S. Fisher
Department of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110
{baekj, fisherp}@wssu.edu

Abstract—We propose a malicious detection algorithm that permits identification of misbehaving wireless stations, and then meting out punishment by not supplying an ACK packet permitting transmission by the malicious stations. The proposed algorithm is designed for IEEE 802.11e network and based upon detecting a change in QoS moving a station up in terms of a measure of permission to a level which is not justified. The impact of non-detection will be for honest stations to set their retransmit attempts to times longer than the deadline set in order to sustain continuous receipt of their data. Our simulation within the ns-2 framework of the IEEE 802.11e EDCA network shows how our algorithm actually detects and adjusts the punishment phase for stations that may be misbehaving as well as stations which are misbehaving.

Keywords—IEEE 802.11e; transmission opportunity; malicious station; cheater detection; cheater punishment

I. INTRODUCTION

Contemporary wireless devices should be equipped with a functionality for supporting Quality of Service (QoS) applications such as voice or video streaming services. Unfortunately, most widely used Medium Access Control (MAC) protocols, IEEE 802.11a/b/g [1], could not satisfy these requirements because they do not provide assurance for fluent QoS. For example, the Distributed Coordination Function (DCF) in the legacy IEEE 802.11 series could not guarantee performance for multimedia applications because it only works as a contention-based channel coordination function. As an alternative, the Point Coordination Function (PCF) has been proposed to provide a contention-free period. During this period, the access point (AP) grants a contention-free channel access to each wireless station with the polling mechanism based upon a round-robin scheme. However, we first note that only one polled station can transmit data in each polling interval [2]. In addition, in AP’s perspective, there is no efficient way to determine the volume of data that will be transmitted from the polled station. This causes unpredictable beacon delay, and it does not allow other stations to fairly access the shared medium to transmit their data. More importantly, it does not include any mechanism to prioritize transmissions among the different data flows.

In order to overcome these limitations, IEEE 802.11e has been proposed by the Task Group e (TGe). Unlike the legacy IEEE 802.11 technologies, IEEE 802.11e considers prioritized traffic using both the contention and contention-free periods for the wireless stations in the super frame. To support two separate periods, TGe has implemented a Hybrid Coordination Function (HCF) consisting of the HCF Controlled Channel Access (HCCA) function and the Enhanced Distributed Channel Access (EDCA) function. For both channel access functions, a new concept, the transmission opportunity (TXOP), has been introduced. That is, during TXOP period, a wireless station can burst its QoS data without any interruption by other wireless stations. For the contention-free period, HCCA is used with the Hybrid Coordinator (HC) installed at the AP. The HCCA defines a traffic specification (TSPEC) frame describing the QoS requirements for each station including maximum and minimum packet size, maximum and minimum data rate, maximum jitter, and the maximum and minimum packet count. Using the TSPEC frame, each QoS enhanced wireless STAtion (QSTA) negotiates with the QoS enhanced Access Point (QAP) for acquiring enough TXOP duration for transmission.

IEEE 802.11e Super Frame

Figure 1 shows structure of the 802.11e super frame which consists of the contention-free period operated by HCCA, and the contention period operated by both HCCA and EDCA. Every super frame starts with the beacon frame which is periodically broadcasted by QAP. The beacon frame includes network parameters which can be used for the contention between wireless stations. The polled TXOP with HCCA can be acquired through the negotiation phase between QSTA and QAP using a TSPEC frame. Once the HC at the QAP determines the TXOP-granted QSTA, it schedules the polled TXOP duration used by the QAP, and informs the QSTA by sending a QoS CF-Poll frame. When the contention-free period ends, QAP broadcasts the CF-End frame to all QSTAs to initiate the contention period with EDCA.

* Supported in part by the National Science Foundation under Award No. 0720856.
On the other hand, the contention period is not managed by any particular mediator in QAP. To transmit data during the contention period with EDCA, each QSTA competes with other QSTAs based upon the broadcasted EDCA parameters. With EDCA, the prioritized flows are considered. That is, it confers a higher probability of transmission of data having a shorter inter-frame space. EDCA categorizes QoS applications with eight user priorities. Usually voice applications have the highest priority, and ordinary best effort applications have the lowest priority. These eight user priorities are grouped into four access categories (ACs). Table I shows mapping relationships between user priorities and ACs. Each QSTA maintains four outgoing packet queues for the four ACs, and each packet would virtually contend with the other packets in other queues having different EDCA parameter values. The EDCA parameters include 1) the arbitrary inter-frame space number (AIFSN[AC]) which is used to calculate AIFS; 2) the minimum value of the contention window (CWmin[AC]); 3) the maximum value of the contention window (CWmax[AC]); and 4) the maximum duration of TXOP (TXOPlimit[AC]). Table II shows the default EDCA parameter values. Note that the smaller values mean there is a greater opportunity to access the channel.

### Possible Misbehaviors

The structure of an EDCA TXOPlimit is shown in Figure 2. Each QSTA can burst its QoS data during the assigned TXOPlimit period and the value of the TXOPlimit can be determined as either a constant or as a dynamic value [3]. We claim all QSTAs must obey the broadcasted EDCA parameter values including TXOPlimit for fair contention among the QSTAs. Otherwise if there is one or more malicious QSTAs which intentionally disrupt fair contention and behave selfishly taking more opportunity to access the medium, other honest QSTAs eventually experience performance degradation over time. Unfortunately, all MAC layer protocols including IEEE 802.11e have been developed assuming all wireless stations will obey the suggested network parameter values honestly. This optimistic assumption is more serious for IEEE 802.11e network because it is designed to implement differentiated traffic. As a result, the malicious QSTAs consume more channel throughput than assigned; and obstruct other QSTAs attempting to transmit their timing-sensitive QoS data. In order to cope with this issue, some important parameters should be measured and examined. In this paper, we first consider possible misbehaviors modifying the parameter values and propose a simple but efficient mechanism to detect the abnormal QSTAs based upon these parameter measurements.

### II. Related Work

In [4, 5], the authors investigate a case of a forged backoff value, and have proposed a new scheme requiring few modifications to the DCF used in the IEEE 802.11a/b/g network. The receiver randomly selects the backoff value based on the lower bound which is assigned by the sender. If the sender’s backoff time is smaller than the assigned value, the receiver considers that the sender is malicious because a smaller backoff time would eventually provide more opportunity to access the shared channel.

Another approach [6] is extended from Bianchi’s model [7]. In order to model the misbehaving wireless stations, it specified some malicious cases where the cheater could fix its contention window. A game-theoretic approach was used to investigate the selfish behaviors with the Nash equilibrium. However, they assumed the network is always in the saturated condition which would be infeasible in the practical situation.

A predictable random backoff (PRB) is proposed in [8]. The key idea of this approach is to adjust the lower bound of the contention window. A 3-D Markov chain is used to compute the system throughput. Although PRB is able to ensure the detection/punishment of malicious/selfish wireless stations, they only consider problematic cases found in mobile, ad hoc networks.

In [9], DOMINO, software to be installed at the access point, is developed. This includes multiple modules to detect various kinds of misbehaviors by wireless stations. However, they could not show cases relevant to EDCA in IEEE 802.11e networks. To the best of our knowledge, there is no literature focusing on the malicious detection issue for IEEE 802.11e network.

### III. Possible Misbehaviors

#### A. Make Shorter AIFS/DIFS/Random Backoff Time

In IEEE 802.11 networks, the malicious/selfish wireless station/QSTA may forge AIFS/DIFS values to minimize the waiting time, or modify AIFS/DIFS+backoff time to transmit its next data packets with a shorter wait interval. As a result, the station/QSTA can increase the probability of channel access by intentionally minimizing the AIFS/DIFS/backoff time. To cope with this problem, we use the approach proposed by [9] with modifications to add the concept of the AC in EDCA.
B. Make A Longer TXOPLimit

One of the most important concepts in the IEEE 802.11e network is TXOP because all QoS data transmissions should be completed within the assigned TXOPLimit to maintain the desirable QoS level of their voice/video streaming applications. As shown in Figure 2, one TXOP cycle consists of DATA and ACK packet pairs with SIFS time. Once a QSTA acquires TXOP duration, other QSTAs cannot intervene during this duration. Therefore, if a malicious QSTA autonomously increases a value of the TXOPLimit, other honest QSTAs must increase their backoff window, potentially missing their deadline to transmit data. We focus on the cases of forging the TXOPLimit by malicious/selfish QSTAs. There are two different methods to determine TXOPLimit value – namely a static and dynamic method. To use the static TXOPLimit, the QAP simply maintains and adjusts the value of TXOPLimit as a constant value, and broadcasts such adjustments to all connected QSTAs. In contrast, the dynamic TXOPLimit can be calculated with the QoS requirements of each QSTA such as lower bounds of throughput, jitter, or delay. The calculation is proposed in reference [3]:

\[
T_i^{\text{required}} = \sum DATA_{dur} \times n_i + (2 \times n_i - 1) \times \text{SIFS} + n_i \times \text{ACK}_{dur},
\]

where \( T_i^{\text{required}} \) is the traffic throughput requirement of QSTA \( i \), \( DATA_{dur} \) is the duration of packet transmission time of QSTA \( i \) based upon the MAC Service Data Unit (MSDU), \( n_i \) is the number of packets of QSTA \( i \), \( \text{ACK}_{dur} \) is the duration to transmit an ACK frame to QSTA \( i \). As SBA is defined as the surplus bandwidth allowance for the expected error performance, the TXOPLimit, for QSTA \( i \) is given by:

\[
\text{TXOPLimit}_i = T_i^{\text{required}} \times \text{SBA}_i.
\]

The example to determine the default SBA value described in TSPEC is available in [10].

IV. POTENTIAL CHEATER DETECTION MECHANISM

We first design and implement the malicious behavior detection algorithm. Our algorithm uses recorded values of the slot time for each QSTA. The QAP records statistics for several beacon intervals. In every beacon interval, the duration of TXOP (TXOPdur) and the inter-frame space size (IFSsize) are recorded.

In order to calculate TXOPdur, the starting time of TXOP and the ending time of TXOP must be measured. The QAP checks the destination address of the previously sent ACK packet whenever it receives a DATA packet. If the current DATA packet’s source address and the previous ACK’s destination address are identical, the QAP recognizes that the TXOP has been started. In order to simplify our discussion, we consider that one cycle of transmission includes DATA, SIFS, ACK, and SIFS. That is, we do not consider the RTS/CTS mechanism, which is not mandatory for the TXOP usage. A TXOP may consist of the multiple cycles. Even though TSPEC contains the size of the DATA packet’s payload defined at the negotiation phase, there could be unexpected link delay by erroneous situations. This consideration is very useful because we can adapt the proposed algorithm to the various bit rate (VBR) data transmissions as well as the constant bit rate (CBR) requirements. In every transmission cycle \( j \) within beacon \( i \), the QAP calculates the actual duration of the DATA packet, DATAdur[i][j], with:

\[
\text{DATAdur}[i][j] = \text{ACKstart}[i][j] - \text{DATAarr}[i][j] - \text{SIFS},
\]

where \( i \) is the current beacon index while \( j \) is the current cycle index. ACKstart is the slot time that QAP sends the ACK packet to the destination. DATAarr is the slot time that the QAP receives the first bit of the DATA packet. DATAdur can include the link delay which is dependent on the DATA payload size described in the TSPEC. Of course, the QAP can calculate duration of one cycle, CYCLEdur, with:

\[
\text{CYCLEdur}[i][j] = \text{DATAdur}[i][j] + \text{SIFS} + \text{ACKdur} + \text{SIFS},
\]

where ACKdur is a constant value determined by the QAP. If the source address of the most recently received DATA packet is different from the address of precisely received packet, the QAP considers that TXOP of the previous one has finished, and starts the detection algorithm. The TXOPdur[i] at current beacon \( i \) can be now calculated by:

\[
\text{TXOPdur}[i] = \sum_{j=0}^{c} \text{CYCLEdur}[i][j],
\]

where \( c \) is the total number of cycles, and \( j \) is the cycle index. Note that the calculated TXOPdur should not exceed the assigned TXOPLimit. Otherwise, the source QSTA can be potentially considered a malicious/selfish node. This simple consideration is not always true because we need to consider a case of consecutive TXOPs. For example, if two consecutive cycles have the same source address, the algorithm recognizes that the measured TXOPdur is much larger than the assigned threshold TXOPLimit, even if those behaviors are legal and normal. Thus, in order to separate two or more consecutive TXOPs, our algorithm has a module to calculate the interframe space size, IFSsize, using:

\[
\text{IFSsize} = \text{currentTime} - \text{ACKstart}[i-1] - \text{ACKdur},
\]

where currentTime is a current slot time index, and ACKstart is a slot time index that QAP sends an ACK packet to the corresponding QSTA. The resultant value IFSsize can be additionally used to determine if the previous TXOP ends or not with:

\[
\text{IFSsize} < \min\{\text{AIFS}[\text{AC}][i][j] \mid 1 \leq i \leq B\},
\]

We consider if the calculated IFSsize is smaller than any defined AIFS value at the current beacon index \( B \), it is considered as SIFS. In other words, the TXOPdur is still increasing. After calculating TXOPdur, the QAP determines if the previous source QSTA is the potential cheater based on:

\[
\text{TXOPdur} \geq \max\{\text{TXOPLimit}[\text{AC}][i][j] \mid 1 \leq i \leq B\} + \sum_{\text{delay}},
\]

where the second term is the cumulated delay including queuing delay, collision, and on-air delay. We assume that the cumulated can be measured and recorded by QAP. Once the QSTA satisfies the condition (8), it is considered as a potential cheater.
V. DETERMINING ACTUAL CHEATER

The next issue is how QAP determines if the potential QSTAs are the actual cheaters. For determination, we propose a penalty-based approach. With our algorithm, we define a flag variable, which is set to true when condition (8) is satisfied. We also define four states of the potential cheater. Figure 4 shows the four states — namely, normal, less suspicious, more suspicious, and punish. We need to mention that our goal is to show how our algorithm operates in multi-stations environment. In actual implementation, the four states should be extended by defining multiple suspicious states if the network administrator wants to allow more misbehavior states. Getting one penalty means moving one transition toward the punish state. If the potential cheater reaches the punish state, the QAP considers it as the actual cheater and does not transmit an ACK packet to it. To use this state transition concept, the QAP needs to record results from the potential cheater detection algorithm over the limited beacon indexes. Figure 5 shows how our penalty function enables transitions between the states. In every pre-determined period of the beacon index, the QAP calls the penalty function which has four input parameters: the current beacon index (bi), the observed time window size (w), size of the selected beacon indexes (s_indexes), and the number of transitions to move forward or backward between the states (transitions).

Our penalty function divides w of the observed beacon indexes into three parts: the latest \([w/2]\) observed beacon indexes, earlier \([w/4]\) observed beacon indexes, and the earliest \(w - ([w/4] + [w/2])\) beacon indexes. If the penalty function finds the flag of the potential cheater within the latest indexes, it gives three points of the penalty to the corresponding potential cheater. In other words, the potential cheater’s state has been changed to the punish state. Likewise, if the penalty function finds the flag within the earlier indexes, it gives two points of the penalty to the potential cheater. And, if the flag is found within the earliest indexes, the penalty function gives one point to the potential cheater.

![Figure 4. State transitions to punish the cheater](image)

![Figure 5. Penalty function](image)

A. Penalty Fuction: Getting The Penalty

If the QAP collects enough link status information of the potential cheater list with beacon indexes, it starts a procedure to add/subtract the penalty point which can be used to determine the actual cheater when running our penalty function at every beacon index.

Our penalty function divides w of the observed beacon indexes into three parts: the latest \([w/2]\) observed beacon indexes, earlier \([w/4]\) observed beacon indexes, and the earliest \(w - ([w/4] + [w/2])\) beacon indexes. If the penalty function finds the flag of the potential cheater within the latest indexes, it gives three points of the penalty to the corresponding potential cheater. In other words, the potential cheater’s state has been changed to the punish state. Likewise, if the penalty function finds the flag within the earlier indexes, it gives two points of the penalty to the potential cheater. And, if the flag is found within the earliest indexes, the penalty function gives one point to the potential cheater.

![Figure 6. Example of the penalty function for node x at the beacon index i](image)

Figure 6 shows an operational example of the penalty function. One of the latest indexes includes the flag of the potential cheater. Thus, the corresponding cheater gets three points of the penalty, and it moves to the punish state. Moreover, one of the earliest indexes also has the flag, the potential cheater gets one more penalty, but it still stays at the punish state.

B. Penalty Function: Losing Penalty

If the penalty function did not find any flag of the potential cheater within the latest indexes, it subtracts three points of the penalty for the corresponding potential cheater. This means that the potential cheater’s state moves back to the normal state. Likewise, not finding the flag within the earlier indexes means subtracting two points of the penalty, and not finding the flag within the earliest indexes means subtracting one point of the penalty. Our proposed penalty function has the advantage of being able to accurately detect the cheater with straightforward extension. We can reduce erroneous detection results only using our potential cheater detection mechanism because our penalty function checks multiple times (triple in our example) based on the previous results of the detection mechanism. We impose more severe penalty to the most recent violation, which has higher impact to the current network condition. Also, our penalty function enables the QAP to use a limited memory capacity per QSTA because the QAP needs to maintain a limited number of results associated with the beacon indexes for the transitions.

VI. SIMULATION

We simulated our proposed mechanism on the NS-2 [11] with an IEEE 802.11e EDCA implementation by TKN [12]. We set a total of five QSTAs. We do not consider the cycle to include RTS and CTS packets since these are not available at the TKN implementation. We determined one beacon interval (one second) as an interval to run our penalty function.

Figure 7 shows a simulation result without the cheater for 30 beacon indexes. There are five QSTAs. The voice application is installed on one QSTA transmitting CBR streaming (timing-sensitive) data through the QAP. The others four QSTAs have video applications. All QSTAs fairly
remains the state much more frequently than other QSTAs. Since, in this suspicious state, Voice returns back to the normal state, QSTA 1, 2, and 4 reach the less suspicious state, QSTA 3 still remains in the normal state. At 14 seconds, Voice returns back to the normal state, QSTA 2 returns back to the more suspicious state, QSTA 3 still remains the normal state, and QSTA 4 returns back to the less suspicious state. As we can see, QSTA 2 reaches the punish state much more frequently than other QSTAs. Since, in this simulation, we do not block the actual cheater to check whether our penalty function works well, the actual cheater still transmits its data, even if it already reaches the punish state. However, if the QAP does not send the ACK packet to the actual cheater right after detecting it, the blocked cheater increases the backoff time, and tries to retransmit. However, at this moment, other honest QSTAs can intercept the channel.

Figure 9. Result from the penalty function with 15 ms cumulated delay

Figure 7. Throughput without cheater

Figure 8. Throughput with cheater

Figure 9 shows a result from our penalty function which executed for 60 seconds by setting the observed window size to 10 seconds (w = 10 seconds). After the 10 seconds, the penalty function runs with the previously observed results. For every one second interval, the penalty function tests if the potential cheater could be an actual one. For example, at 11 seconds, the penalty function determines that Voice reaches the less suspicious state, QSTA 1, 2, and 4 reach the punish state, and QSTA 3 still remains in the normal state. At 14 seconds, Voice returns back to the normal state, QSTA 2 moves back to the more suspicious state, QSTA 3 moves back to the less suspicious state, and QSTA 4 returns back to the more suspicious state. The results from the potential cheater detection mechanism, the penalty function runs to detect and block the actual cheater among of the detected potential cheaters.

VII. CONCLUSION

In the IEEE 802.11e EDCA network, the malicious stations prevent other honest stations access the channel. To detect the potential cheaters, we proposed the potential cheater detection mechanism which enables to find out the stations maliciously increasing TXOPLimit values. After cumulating the results from the potential cheater detection mechanism, the penalty function runs to detect and block the actual cheater. The penalty function has the four states, normal, less suspicious, more suspicious, and punish. The QAP does not send the ACK packet to the actual cheater when it reaches the punish state. The results from the simulation show our proposed mechanism detects the actual cheater efficiently in real-time. We will further our study to conclude how the number of states and the size of beacon index group affect to find the actual cheaters.

REFERENCES