PERFORMANCE ENHANCEMENT OF THE EDCA PROTOCOL FOR THE CHANNEL ACCESS IN NEXT GENERATION IEEE 802.11 AX WLAN

by

MD. MAHMUDUL HASAN

MASTER OF SCIENCE IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

2019

The thesis titled "PERFORMANCE ENHANCEMENT OF THE EDCA PROTOCOL FOR THE CHANNEL ACCESS IN NEXT GENERATION IEEE 802.11 AX WLAN" submitted by MD. MAHMUDUL HASAN, Roll No.: 0412312028P, Session: APRIL, 2012 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Science in Information and Communication Technology on 18 March, 2019.

CANDIDATE'S DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Signature of the Candidate

[illegible]

MD. MAHMUDUL HASAN

DEDICATION

I dedicate this thesis to my family members.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Abbreviation of Technical Terms

AC	Access category
AP	Access Point
ACK	Acknowledgement
AIFS	Arbitration Interframe Space
AV	Audio Video
BEB	Binary Exponential Backoff
BSS	Basic Service Set
CAP	Controlled Access Phase
CBSA	Credit Based Shaper Algorithm
CCA	Clear Channel Assessment
CFP	Contention Free Period
CMSE	Cumulative Mean Squared Error
CP	Contention Period
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear to Send
CW	Contention Window
DCF	Distributed Coordination Function
DIFS	Distributed Interframe Space
EDCA	Enhanced Distributed Channel Access
EDCAF	Enhanced Distributed Channel Access Function
EIFS	Extended Interframe Space
GATS	Group Address Translation Service
GoP	Group of Pictures
FCS	Frame Check Sequence
HC	Hybrid Controller
HCF	Hybrid Coordination Function
HCCA	HCF Controlled Channel Access
IEEE	Institute of Electrical and Electronics Engineers

IoT	Internet of Things
MAC	Medium access control
MPEG	Moving Picture Experts Group
MSDU	MAC Service Data Unit
MU-MIMO	Multi User Multiple Input Multiple Output
NAV	Network Allocation Vector
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
PC	Point Coordinator
PCF	Point Coordination Function
PIFS	PCF Interframe Space
PHY	Physical Layer
QAM	Quadrature Amplitude Modulation
QoE	Quality of Experience
QoS	Quality of Service
QAP	QoS AP
QSTA	QoS STA
RTS	Request to Send
SIFS	Short Interframe Space
SPA	Strict Priority Algorithm
SNR	Signal to Noise Ratio
STA	Station
SQPS	Single Queue Priority Scheduler
TC	Traffic Class
TS	Traffic Stream
TWT	Target Wake Time
TXOP	Transmission Opportunity
UHD	Ultra High Definition
VCH	Virtual Collision Handler
VNI	Visual Networking Index
VoIP	Voice over Internet Protocol

WCBSA	Wireless Credit Based Shaper Algorithm
WLAN	Wireless Local Area Network
WM	Wireless Medium

List of Symbols

AC_DVO	Access Category for HD Voice
AC_VO	Access category for Voice
AC_KVI	Access category for 4K and upper resolutions Video
AC_DVI	Access category for 2K and upper resolutions Video
AC_VI	Access category for 1K and lower resolutions Video
℘k	Queue represents AC_KVI
℘d	Queue represents AC_DVI
℘i	Queue represents AC_VI
Ֆd	Queue represents AC_DVO
Ֆo	Queue represents AC_VO

Acknowledgement

I am very grateful to my thesis supervisor Dr. Mohammad Arifuzzaman for giving me an opportunity to complete my thesis work by sharing his ideas and providing me a brilliant guidance throughout the whole Thesis workout. Also I am very much beholden to the Director Professor Dr. Md. Saiful Islam and other faculty members for cooperating me during my research period and many other intermediate tasks to finish the thesis. I am also indebted to the officers and stuffs of the IICT who gave me full support regarding various processing of my thesis lifecycle.

Abstract

For supporting Quality of Service (QoS) functionality and differentiated access to the wireless local area network (WLAN) Enhanced Distributed Channel Access (EDCA) protocol was developed in IEEE 802.11e amendment which is the enhancement to the traditional Medium Access Control (MAC) protocol. This protocol sets priority of Audio and Video (AV) traffic over other traffic in the network. Later, a new amendment called IEEE 802.11aa has been introduced which modifies the EDCA protocol using alternate queues for making AV transmission more efficient. Now a days, online streaming of Ultra High Definition (UHD) videos are getting much popular to the users for enhanced user experience. Also, day by day several types of video standards (i.e. 3K, 4K, 8K etc.) are introduced, which have higher resolutions and differential frame rates. But the conventional protocol treats all the video traffic in the same manner which may lead to the unexpected performance degradation of the real time UHD videos. Viewers of online AV streams may experience huge lag in the real time. The amendment of IEEE 802.11aa which was proposed in 2012 also become obsolete and cannot effectively handle QoS support for time critical video traffic. It creates a demand for new way to treat the video traffic in the wireless medium. Hence, we propose the protocol Additional Video Access Queue Based EDCA (AVAQ-EDCA) to address the said issues. Recently, we have got much attention that a new amendment called IEEE 802.11ax is expecting to be announced on 2020 by making big change in Physical and MAC layer features (e.g. introducing OFDM, Spatial Reuse etc.) which will make faster data transmission in the network than ever. Moreover, in such huge contending situation there must need a suitable MAC protocol which can give priority access to UHD video traffic by which we can overcome the unexpected delay of the time critical traffic. In this thesis, by introducing additional queue we restructured the queueing framework of the IEEE 802.11 aa in order to provide a biased priority access of channel for UHD video traffic. Finally, a simulation model based on NS3 network simulator to compare the performance of our proposed AVAQ-EDCA technique with the conventional EDCA mechanism. We also compare our protocol with Robust Wireless Credit Based Shaper Algorithm (RWCBSA) which is a newly developed MAC algorithm based on IEEE 802.11aa alternate queues formula to transmit frames. Simulation results show that our protocol outperforms both conventional EDCA and RWCBSA.

Chapter 1

Introduction

When wireless devices become available then people take it as their company for daily life and started communicating everywhere through it. Now a man cannot imagine a minute without a wireless smart device. As with the internet service become more speedy people started to use it for voice and video communication in real time. We can see that at first, MAC [1] service was introduced to WLAN to provide contention based channel access to the devices of a Wireless Network. Later, with increasing the efficiency of the WLAN network and availability of various multimedia service in online there need a new model for channel access. For accessing multimedia services efficiently IEEE 802.11e [2] for WLAN was proposed which gives 4 types of differentiated access to the data of the medium and set higher priority to access the AV data to the channel. As the real time streaming over the wireless network become more popular and increasing the demand for the intelligent network needed much to handle real time audio and video traffic. Then a amendment called IEEE 802.11aa [3] announced aim to give robust audio video transmission over the network. Now a days the smart devices become increasing rapidly with a higher growth rate than any services to the human. In different places such as wireless corporate office, outdoor hotspots, dense residential apartments, railway stations, bus stop, airports, parks and stadiums where creates high demand for numerous amount of data transfers for HD/Full HD/3K/4K/8K video, High Density-High Definition collaboration, and Internet of Things (IoT). For the increasing availability of the technology every man is taking the opportunity to watch any event by real time streaming which gives them a lot of enjoyment. To meet up the need for high demand it is necessary to enhance the network capacity. Considering such dense usage scenario a new amendment of WLAN is waiting for announcement by 2020 is IEEE 802.11ax [4-6]. Here, we have seen so many new technology adopted for MAC and Physical layer. Moreover, we have not found any proposal by which the prioritize video traffic can access the channel over the other video categories. That’s why, we have focused on improving the channel access procedure of differential video traffics in such overloaded networks to find better solution for this problem.

1.1 Motivation

In a network there are various types of data for different purposes. Some services are time critical and some are not. In 2015, cisco issued a Visual Networking Index (VNI) at [7] that predicted increase in video traffic across the Internet from 64% in 2014 while it being 75 percent in 2017 where total internet traffic consumed 122 petabytes per month and it will be 82% expected in 2022 where the internet traffic will be consumed 396 petabytes per month which stated at [8]. Now a days everyone is using voice over internet protocol (VoIP) service and real time online video streaming service every time. That’s why the data over the wireless network is increasing so much. And end user of the network is experiencing latency in viewing the AV services. For this reason, now QoS provisioning is a burning issue for WM. As, we need to find a proper mechanism for the proper access mechanism for the differential types of AV traffic. Although it is a critical task to satisfy the end user of the network but if we can improve channel access mechanism for time critical priories AV traffic only then we can improve the visual experience of the end user. Recently, we have taken a look on the techniques has been proposed for the varying demand of traffic. As today the network size, devices and demand everything going increasing, we have tried to find a way how to meet the needs of the upcoming High capacity and High collaboration network. We have reviewed some existing work in for QoS provisioning in the WLAN and address some limitations for the future networks. Thus, we think for reorganize the EDCA mechanism to make an efficient channel access procedure which can lead to the way of improving the existing way of channel access of AV traffic. As in IEEE 802.11ax lets access points support more clients in dense environments and to provide a faster transmission service we can create more one queue for priority video access to the channel, which provide a much better solution for the priories video traffic.

1.2 QoS Parameters

QoS is defined as the measure of performance for a transmission system that reflects its transmission quality and service availability. Service availability is an important fundamental element of QoS. Implementation of QoS is depend on the highly availability of the network infrastructure. The network quality is determined by the following factors in [9-10]:

i. *Latency* is the amount of time it takes a packet to reach the receiving endpoint after being transmitted from the sending endpoint. This time period is generally termed as "end-to-end delay" and can be classified as:

a) *Fixed network delay* includes encoding/decoding time (for voice and video), as well as the amount of time required for the electrical or optical pulses to traverse the media in the route to their destination.

b) *Variable network delay* generally refers to network conditions, such as congestion, that may affect the overall time required for transit.

ii. *Jitter (or delay-variance)* is the difference in the end-to-end latency between packets. For example, if one packet requires 100 milliseconds (ms) to traverse the network from the source-endpoint to the destination-endpoint and the other packet requires 125 ms to make the same trip, then the jitter is calculated as 25 ms.

iii. *Loss (or packet-loss)* is a comparative measure of packets correctly transmitted and received to the total number that were transmitted. Loss is expressed as the percentage of packets that were dropped.

1.3 Challenges in current 802. 11 Networks

Challenges addressed by current 802. 11 Networks addressed by [11] are:

(i) Lack of differentiation among AV streams.

(ii) Lack of mechanism for graceful degradation of AV stream quality.

(iii) Interference caused due to large scale implementation.

(iv) Collision increases due to contention different types of frames.

(v) Lack of reliable and scalable mechanism.

1.4 Research Objectives

The goal of this research work is to give high priority access to the high definition traffic in the network which lead to the efficient channel access of the time critical traffic and crates a better service than before.

- To develop an efficient multiple Access queue based EDCA protocol to resolve the medium access problem for UHD and time sensitive traffic of Next Generation 802.11ax wireless network.
- To devise a way which will lead us to the efficient channel access in dense IEEE 802.11 ax network.
- To develop a new priority mechanism which will efficiently map UHD video and time sensitive traffic in the Next Generation IEEE 802.11ax WLAN network and evaluate the performance of the proposed protocol.

1.5 Outline of Methodology

The methodology consists of the following stages:

- After studying existing related work, a novel protocol with multiple access queue based algorithm will be proposed which will support a better solution for the contention access of UHD video and time sensitive traffic in the Next Generation IEEE 802.11 ax networks.
- Our multiple access queue consists of eight access categories which will map traffic based on their types.
- A simulation model based on NS3 simulation framework will be implemented to evaluate the performance of the proposed protocol and compare with the existing protocols in terms of delay and throughput.

1.6 Organization of Thesis

The subsequent parts of the thesis are organized as follows:

Chapter 2: Literature Review gives an overview of this thesis work covers a detail view of IEEE 802.11 standard of WLAN and also IEEE 802.11e standard. MAC mechanism of IEEE 802.11aa and IEEE 802.11ax and is also discussed in this chapter. *Chapter 3: Proposed protocol* presents a modification to the EDCA access procedure to enhance the QoS of the Audio, Video in the wireless network is discussed in details. *Chapter 4: Simulation Results presents an investigation of* the performances among proposed technique and a recently invented method with legacy EDCA approach in terms of performance parameters such as Mean delay, Packet delivery ratio throughput etc. *Chapter 5: Conclusion and Future Work concludes the thesis by summarizing the main ideas and some directions for future research.*

1.7 Conclusion

This chapter represents a very brief introduction of WLAN. A short description about QoS parameters of the WLAN is described here. Motivation and research methodology are also mentioned here to get an overview of the outcome of this research work. Finally, organization of the thesis is presented.

Chapter 2

Literature Review

2.1 MAC Architecture

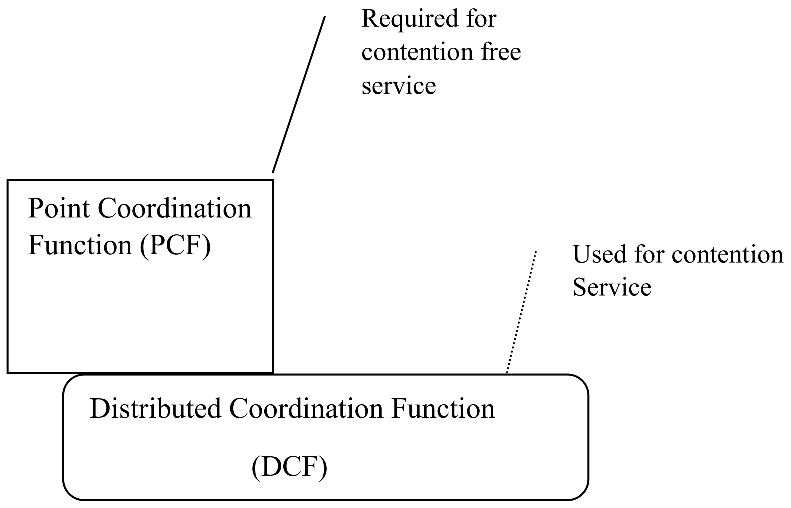
Required for contention free service
Point Coordination Function (PCF)
Used for contention Service
Distributed Coordination Function (DCF)

Figure 2.1: IEEE 802.11 MAC Architecture

2.1.1 Distributed Coordination Function (DCF)

The fundamental access method of the IEEE 802.11 is a DCF which is known as carrier sense multiple access with collision avoidance (CSM/CA). This protocol is used in wireless networks because they cannot detect the collision so the only solution is collision avoidance. From [12-13] we get the DCF procedure as follows:

At first, the station (STA) listen to the medium if it is idle or not? After detecting that there is no other transmission in progress on the WM STAs deliver MAC Service Data Units (MSDUs) of arbitrary lengths. If two stations detect the channel as free at the same time, a collision occurs. The 802.11 defines a Collision Avoidance (CA) mechanism to reduce the probability of collisions. That is, before starting a transmission a station performs a backoff procedure. It sense the channel for an additional random time after detecting the channel as being idle for a minimum duration called DCF Interframe Space (DIFS), the duration is between 28 to 50 μs. Only if the channel remains idle for this additional time period, the station is allowed to start the transmission.

$$DIFS = SIFS + (2 \times Slot\ Time).\ (2.1)$$

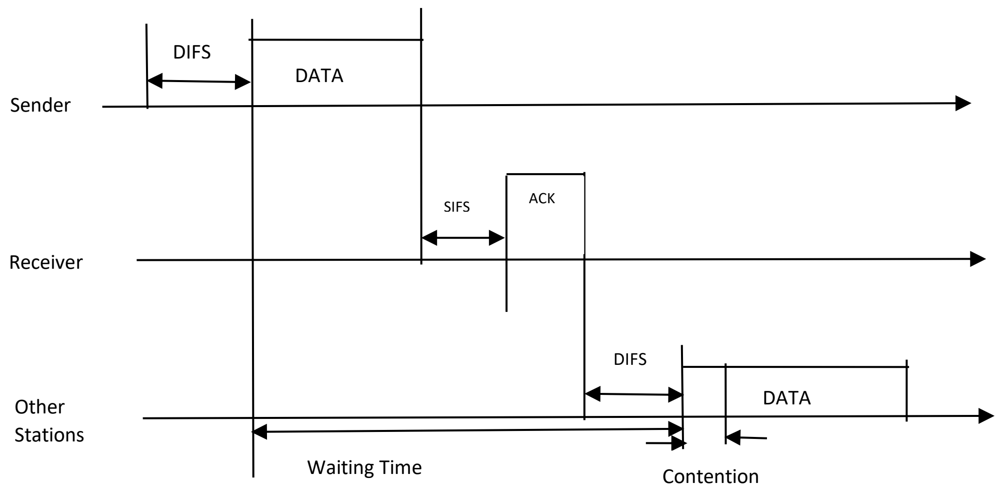

Figure 2.2: Basic DCF Mechanism

Where, the slot time is the time it takes to pass the data through the maximum length of the network. For wireless networks, this depends on the standard used and is between 9 μs and 20 μs. Each STA has a Contention Window (CW), which is used to determine the number of slot times a station has to wait before transmission. A transmitting STA when sense idle channel for a time period of DIFS then it generates a random backoff timer chosen uniformly from the range [0, w − 1], where w is referred to as the contention window or backoff window. At the first transmission attempt, w is set to CWmin (minimum contention window). When the backoff timer reaches to 0, the STA transmits a RTS or data. The backoff window is initially 0. The backoff counter is decreased by 1 for every slot, as long as the channel is sensed idle in that slot. If there are transmissions by other stations during the slot, then the station freeze its backoff counter and will resume the count from where it freezes previous time, after the other transmitting STA has completed its frame transmission and an additional DIFS interval. If the transmission become successful, then the backoff counter is start again. Upon an unsuccessful transmission, the backoff counter is increasedby 1, and the backoff counter is chosen again between 0 and CW for the new backoff stage. The CW value for the next backoff stage is doubled, until it reaches CWmax (contention window maximum). The value of Cwmin in 0, and CWmax is 1023 in legacy DCF. This CW reduces the collision probability in case where there are multiple stations attempting to access the channel. CW is that a period of time in which the network is operating in contention mode. It bounds the range of the generated random backoff timer. The backoff timer is decreased as long as the medium is sensed to be idle for a DIFS, and frozen when a transmission is detected on the medium, and resumed when the channel is detected as idle again for a DIFS interval. When the backoff reaches 0, the station transmits it packet. The backoff length is calculated as follows:

$BackoffLength = Random(0, CW) \times aSlotTime.$ (2.2)

After an unsuccessful transmission the contention window is updated as follows:

$CW = 2^{2i} - 1.$ (2.3)

Where, i is the number of transmission attempts.

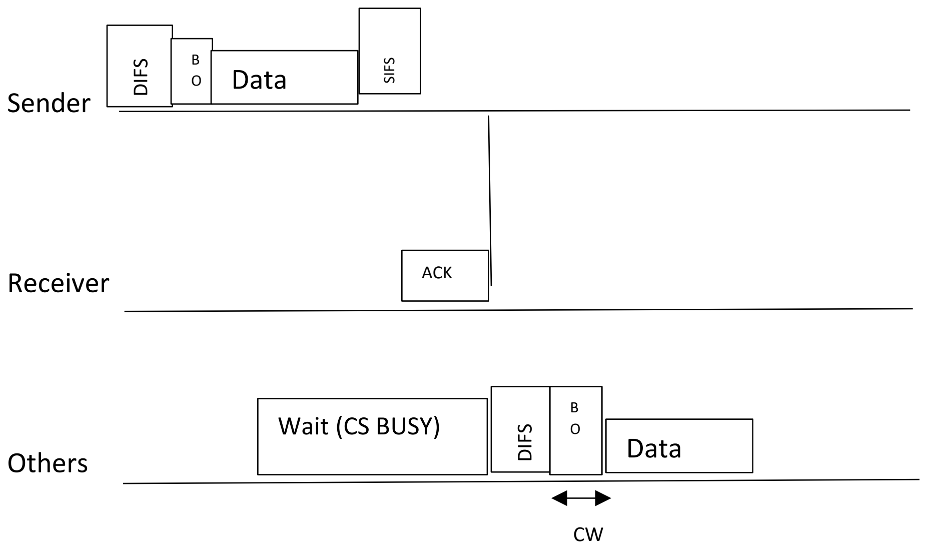

Figure 2.3: Backoff and Contention Window

The Short Inter frame Space, SIFS is defined in the refinement of this method when used request to send (RTS) and clear to send (CTS) and can be used optionally. This method can be used for the issue (i.e. hidden node problem) to further minimize collisions where the transmitting and receiving STA exchange short control frames known as after determining that the medium is idle after any backoff period, prior to data transmission. Before transmitting data frames, a station has the option to transmit a short RTS frame, followed by the CTS transmission by the receiving station. The RTS and CTS frames include the information of how long it does take to transmit the next data frame, i.e., the first fragment, and the corresponding Acknowledgement (ACK) response. Thus, other stations close to the transmitting station and hidden stations close to the receiving station will not start any transmissions for this period of time Network Allocation Vector (NAV).

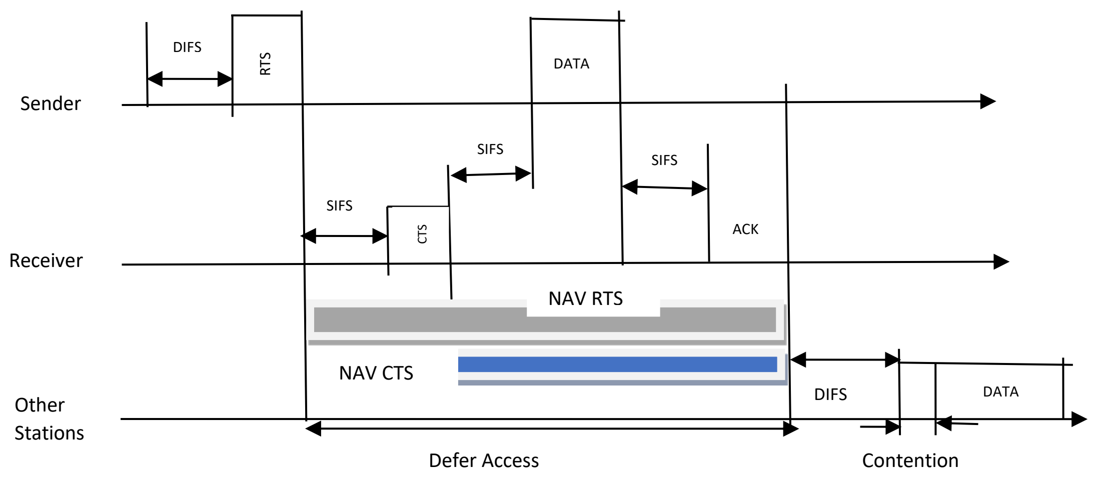

Figure 2.4: DCF with RTS/CTS

After receiving the frame, the receiving STA to respond with an ACK), generally an ACK frame, if the frame check sequence (FCS) of the received frame is correct. This technique is known as positive acknowledgment. Lack of reception of an expected ACK frame indicates to the source STA that an error has occurred.

The Extended Inter frame Space (EIFS) is used by the DCF when the PHY indicated to the MAC that a frame transmission was begun that did not result in the correct reception of a complete MAC frame with a correct FCS value. Reception of an error-free frame during the

EIFS resynchronizes the STA to the actual busy/idle stage of the medium, so the EIFS is terminated and normal medium access procedure continues to receipt frame.

2.1.2 Point coordination function (PCF)

The PCF [14-15] is additionally integrated into the CSMA/CA procedure and supplements the DCF. The access point (AP) acts as a point coordinator (PC) and specifically addresses the stations within the network. The coordinator has a polling list for the order of the individual stations. According to this list, each network participant is asked in order of priority whether they want to transmit or not? Before the access point starts the request, however, it has to wait for a certain time just like DCF. The PCF interframe space (PIFS) is one slot time shorter than the DIFS and has a higher priority, so PCF takes effect earlier than DCF. The method offers an approach to solve the hidden station problem: the required range can be halved if the access point is well positioned. PCF means that participants no longer have to recognize each other. The access point just needs to be placed in the middle and can reach all stations in a star shape.

However, all the participants need to be PCF activated for using this technology. Which devices are not participating in the PCF process, their transfer requests is ignored. For that an alternating system has been developed: PCF and DCF can alternate to give all devices in the network the possibility of transmission. The access point provides two time periods for this. Firstly, there is the contention free period (CFP), in which PCF ensures coordinated multiple access. During the CFP, the PCF is used for accessing the medium, while the DCF is used during the CP. It is necessary that a super frame includes a CP of a minimum length that allows at least one MSDU delivery under DCF. A super frame called beacon frame is a management frame that maintains the synchronization of the local timers in the stations and delivers protocol related parameters. The PC, generates beacon frames at regular intervals, thus every station knows when the next beacon frame will arrive. See in fig 2.5 the PC polls a station asking for a pending frame. Because the PC itself has pending data for this STA, it uses a combined data and poll frame by piggybacking the CF-Poll frame on the data frame. After polled, along with data, the polled station acknowledges the successful reception. If the PC received no response from a polled station after waiting for a PIFS, it polls the next station, or ends the CFP. Thus no idle period longer than PIFS occurs during CFP.

$$\boldsymbol{PIFS = a\,SIFSTime + a\,SlotTime} \quad .(2.4)$$

The PC continues with polling other stations until the CFP expires. A specific control frame, called CF-End, is transmitted by the PC as the last frame within the CFP to signal the end of the CFP. Despite the support of this mechanism in the 802.11 standard, IEEE left some details unaddressed, such as scaling PCF in an enterprise network, which left PCF as an immature access method. Vendors never saw interest to meet this development in practice, and as a result, PCF has gone unused in 802.11 WLANs.

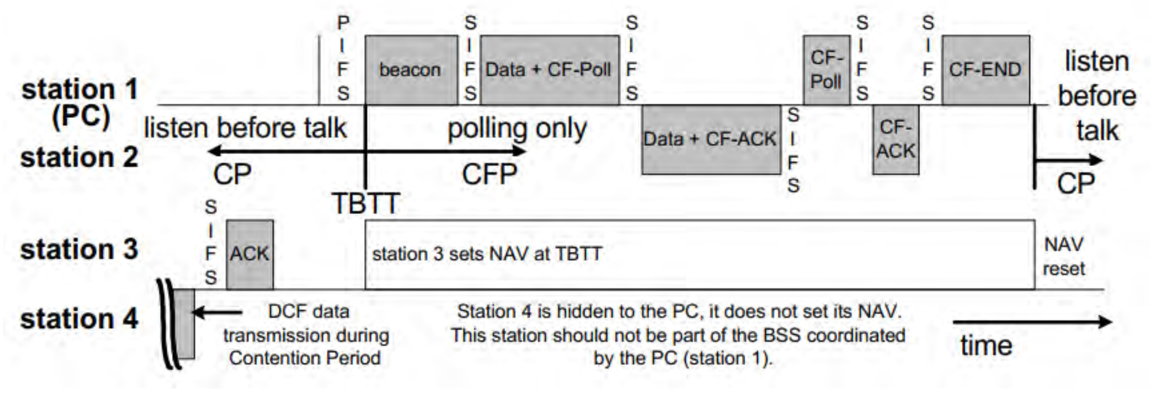
station 1 (PC)
station 2
station 3
station 4
PIFS
beacon
SIFS
Data + CF-Poll
SIFS
Data + CF-ACK
SIFS
CF-Poll
SIFS
CF-ACK
SIFS
CF-END
listen before talk
polling only
CP
CFP
TBTT
listen before talk
CP
SIFS
ACK
station 3 sets NAV at TBTT
NAV reset
DCF data transmission during Contention Period
Station 4 is hidden to the PC, it does not set its NAV. This station should not be part of the BSS coordinated by the PC (station 1).
time

Figure 2.5: PCF operation. Adopted from [14]

We can see from the fig 2.5 above that the Station 1 is the PC which polling station 2. Station 3 detects the beacon frame and sets the NAV for the whole CFP. Station 4 is hidden to station 1 and does not detect the beacon frame; it continues to operate in DCF.

2.2 Hybrid Coordination Function (HCF)

As an optional access method in addition to DCF, HCF was introduced to support QoS. HCF joints the elements of both DCF and PCF mechanisms, creating a contention-free HCF method, called HCCA and a HCF method with contention access is called EDCA.

The HCF need to implement in all QoS suppoted STAs (QSTAs). It combines functions from the DCF and PCF with some enhancement, QoS-specific mechanisms and frame subtypes to allow a uniform set of frame exchange sequences to be used for QoS data transfers during both the CP and CFP.

QSTAs may obtain Transmission Opportunity (TXOP) using one or both of the channel access mechanisms. If a TXOP is obtained using the contention-based channel access, it is defined as EDCA TXOP. If a TXOP is obtained using the controlled channel access, it is defined as HCCA TXOP. If the HCCA TXOP is obtained by a QoS + CF-Poll frame from the HC, the TXOP is called a polled TXOP.

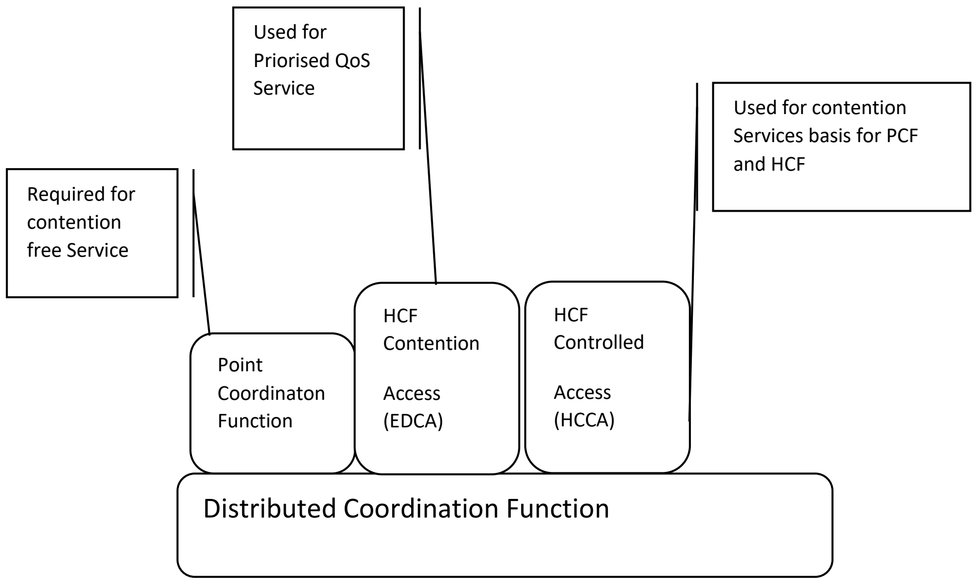
Used for Priorised QoS Service
Used for contention Services basis for PCF and HCF
Required for contention free Service
Point Coordinaton Function
HCF Contention Access (EDCA)
HCF Controlled Access (HCCA)
Distributed Coordination Function

Figure 2.6: MAC Architecture with HCF

2.2.1 HCF controlled channel access (HCCA)

The HCCA works a lot like PCF. However, in contrast to PCF, in which the interval between two beacon frames is divided into two periods of contention-free period (CFP) and contention period (CP), the HCCA allows for CFPs being initiated at almost any time during a CP. Such kind of CFP is called a Controlled Access Phase (CAP). A CAP is initiated by the AP when it wants to send a frame to a QSTA or receive a frame from a QSTA in a contention-free manner. During a CAP, the Hybrid Coordinator (HC), which is an AP called QoS aware AP (QAP), controls the access to the medium. During the CP, all stations function in EDCA. The other difference with the PCF is that Traffic Class (TC) and Traffic Streams (TS) are defined. This means that the HC is not limited to per-station queuing and can provide a kind of per-session service. Also, the HC can coordinate these streams or sessions in any fashion it chooses. Moreover, the stations give info about the lengths of their queues for each Traffic Class (TC). The HC can use this info to give priority to one station over another, or better adjust its scheduling mechanism. Another difference is that QSTAs are given a TXOP they may send multiple packets in a row, for a given time period selected by the HC. During the CFP, the HC allows stations to send data by sending CF-Poll frames.

HCCA is generally considered the most advanced and complex coordination function. With the HCCA, QoS can be configured with great precision. QSTAs have the ability to request specific transmission parameters (data rate, jitter, etc.) which should allow advanced applications like VoIP and video streaming to work more effectively on a Wi-Fi network. HCCA support is not mandatory for 802.11e APs. In fact, few APs currently available are enabled for HCCA Implementation with the existing DCF mechanism for channel access.

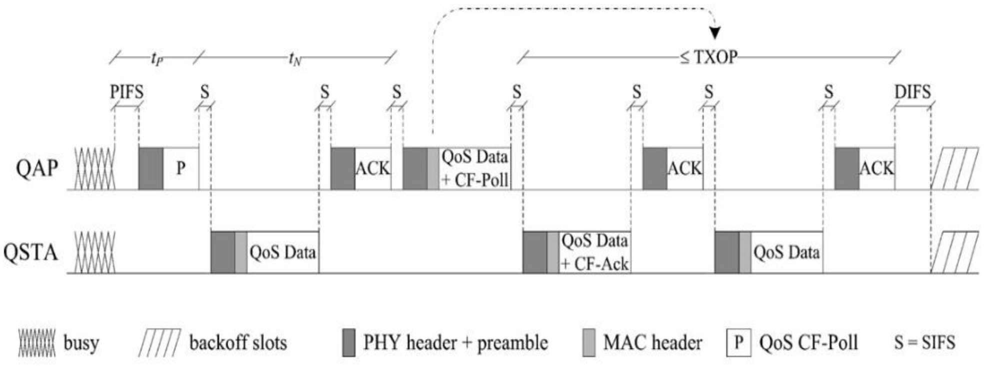

Figure 2.7: HCCA sample frame exchange (Adopted from [16])

2.2.2 Enhanced Distributed Channel Access (EDCA)

This is a contention-based channel access method which priory's contention-based wireless medium (WM) access by classifying 802.11 traffic types by User Priorities (UP) and Access Categories (AC). There are a total of 8 UPs, which map to 4 ACs, as shown in table below.

Table 2.1: User priority Mappings to Access Category in IEEE 802.11e

User Priority	Priority	Access Category(AC)	Designation
1	Lowest	AC_BK	Background
2	↓	AC_BK	Background
0		AC_BE	Best Effort
3		AC_BE	Best Effort
4		AC_VI	Video
5		AC_VI	Video
6		AC_VO	Voice
7	Highest	AC_VO	Voice

For each AC, an enhanced variant of the DCF, called an Enhanced Distributed Channel Access Function (EDCAF), contends for TXOPs using a set of EDCA parameters from the EDCA Parameter Set element or from the default values for the parameters when EDCA Parameter Set element is received from the QAP with which the QSTA is associated. EDCA regulates the access of wireless channel by EDCAF [13]. A station may run up to 4 EDCAF's, and each frames generated by the station is mapped to one of these EDCAF's. Then each EDCAF executes an independent backoff process to transmit its frame. EDCA make possible the differentiated access through EDCA parameter set AIFS[AC], CWmin[AC], CWmax[AC], and TXOPlimit[AC] for a corresponding AC (N) [17]. A Contention Access Function (CAF) wants to transmit a frame monitors the channel activity. In EDCA, every AC behaves as a single station. It waits for a time AIFS [AC] which is defined as:

$$AIFS[AC] = SIFS + AIFSN[AC] \times SlotTime\ . \quad (2.5)$$

Where, AIFSN[AC] = 2 to 7, It is an integer indicating the number of slots after a SIFS duration a station should defer before either invoking a backoff or starting a transmission. The minimum value for AIFSN is two for backoff entities of a QSTA and one for back entities of QAP. Slot Time is the duration of one slot. The backoff entities are prioritized according to the values of their EDCA parameter sets. The smaller the AIFS [AC] or CWmin[AC], the higher the priority in medium access, and thus, the higher the throughput. The backoff interval for an AC in EDCA is randomly selected from [1, CW], instead of [0, CW −1] as in DCF. The operation of 802.11e EDCA is described as follows [18]:

When each of the data frame from the higher layer arrives in the MAC layer with a specific priority value. Then, each frame is mapped into an AC with specified priority. The purpose of using different contention parameters for different queues is to give a low priority class a longer waiting time than a high-priority class, so the high-priority class is likely to access the medium earlier than the low-priority class. The values of the EDCA parameter set for each AC are announced periodically by the QAP via beacon frames. Each AC behaves as a single EDCAF, and the corresponding queue has its own AIFS, backoff interval, and CW. If the channel is idle for a period of time equal to the AIFS [AC], the CAF transmits. After each unsuccessful transmission attempt, the contention window is doubled until a retry limit or the maximum contention window is reached. The collision is handled in a virtual manner. That is, the highest priority frame among the colliding frames is chosen and transmitted, and the others perform a backoff with an enlarged CW value.

Collisions between contending EDCAFs within a QSTA are resolved within the QSTA so that the data frames from the higher priority AC receives the TXOP and the data frames from the lower priority colliding ACs behave as if there were an external collision on the WM. During an EDCA TXOP won by an EDCAF, a QSTA may initiate multiple frame exchange sequences to transmit MMPDUs and/or MSDUs within the same AC. The duration of this EDCA TXOP is bounded, for an AC.

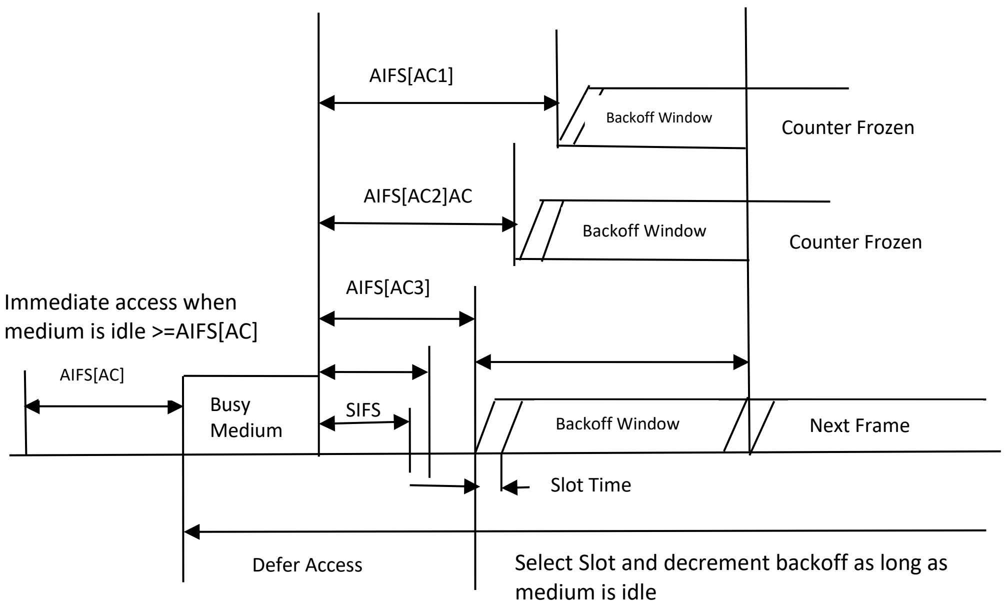

Figure 2.8: IEEE 802.11e EDCA Operation

Table 2.2: EDCA Parameter Particulars

Access Category	AIFS	CWmin	CwMax	TXOP Limit
AC_VO	2	3	7	2.080
AC_VI	2	7	15	4.096
AC_BE	3	15	1023	0
AC_BK	7	15	1023	0

A station that wins an EDCA contention is granted a TXOP which is the right to use the medium for a period of time. The duration of this TXOP is specified per access category, and is contained in the TXOP limit field of the access category (AC) parameter that is recorded in the EDCA parameter set. A QoS STA can use a TXOP to transmit multiple frames within an access category. Each of the 4 ACs has a different TXOP value. The AC_BK and AC_BE categories have a TXOP value 0. This means these can only send 1 frame during their TXOP. Once that frame it sent, it must contend for the WM again with the CCA, NAV, AIFS, and CW. The AC_VI has a TXOP value of 4.096ms. This means the video AC can send as many frames it can within the time it has been granted. Once its TXOP is up, it must contend for the wireless medium again if it has additional frames to send. In a similar way AC voice send traffic with a TXOP limit of 2.080ms.

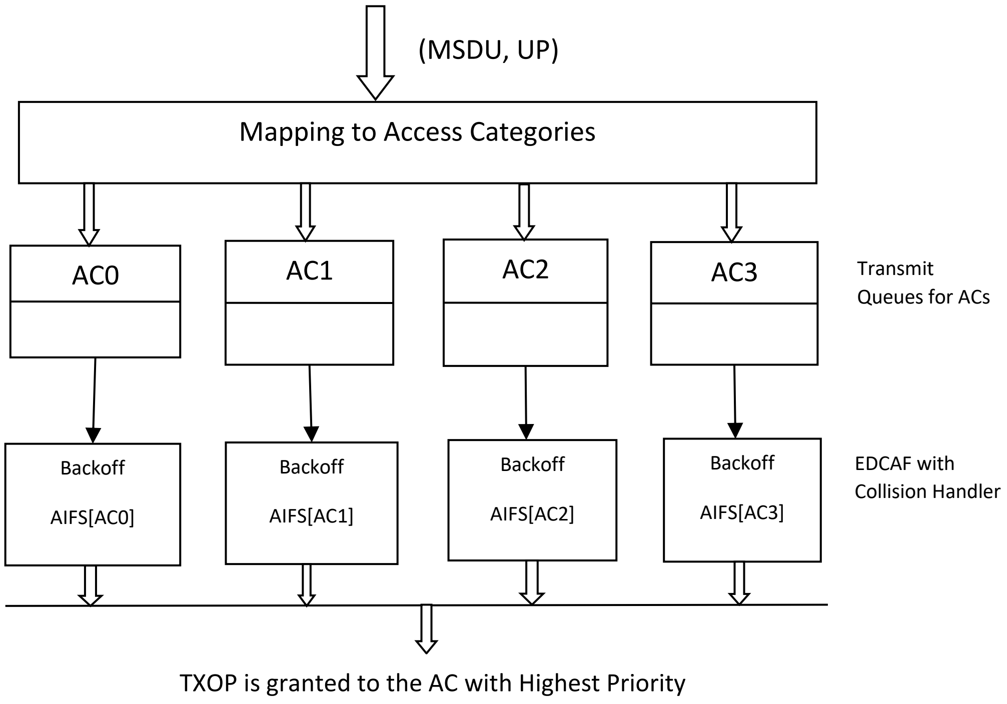

Figure 2.9: Internal mechanism of EDCA

According to the standard stated in [19], when a TXOP is not successfully accessed or the data frame is dropped, due to exceeding the retry limit, the station has to perform the zeroth backoff phase count. It performs the minimum CW size and zero value for retry count, regardless of whether there is a data frame in the queue or not . When the backoff counter reaches to zero, the station checks if there is a pending frame in the queue or not.

In the figure 2.10 below STA1 wants to send data first listen for the medium for AIFS[3] which is 3 slot time (from table 2.2) sending best effort (AC3) data. Then it send RTS after a random backoff to and get CTS from the receiver (here STA2). Other STA's sets NAV and freeze the backoff counter. After successfully transmits data receiver send ACK to the STA1. Then it again starts AIFS[3]. Meanwhile, STA3 with voice data starts AIFS[1] which is 2 time slots. That's why STA1 finds the medium is idle and send RTS to allocate channel. STA1 get the allocation notification and freezes its bakcoff counter. That's how the priority based AC get the TXOP faster.

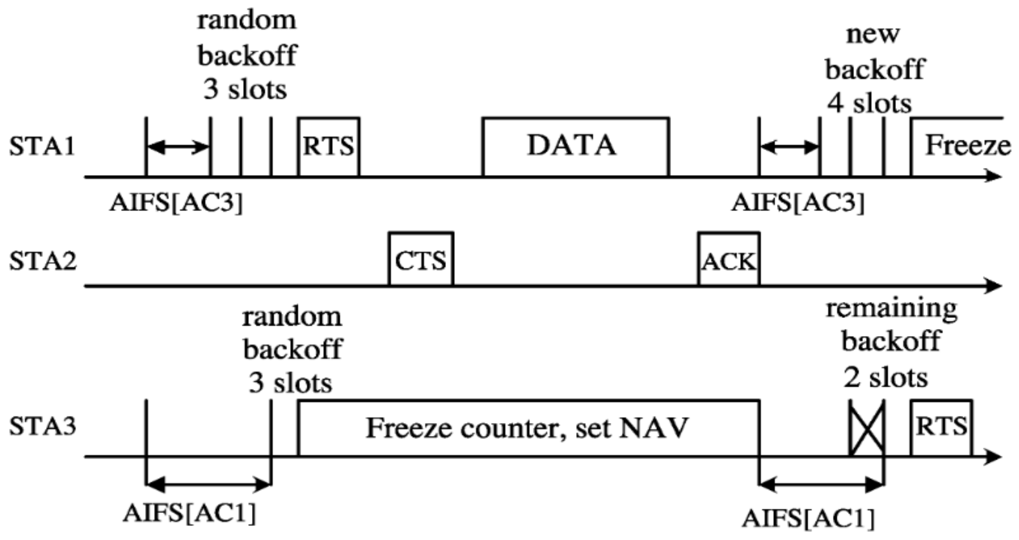

Figure 2.10: EDCA access mechanism (Adopted from [20])

2.3 MAC Enhancement for Robust Audio Video Streaming

Later a modification of convention EDCA model came out in 2012 which is discussed in below:

2.3.1 IEEE 802.11aa

Intra-access category prioritization [3] provides six transmit queues that map to four enhanced distributed channel access functions (EDCAFs) to enable differentiation between traffic streams that are in the same access category in order for finer grained prioritization to be applied between individual AV streams. This mechanism is developed in IEEE 802.11 aa amendment for robust audio video transmission. Priority mappings of IEEE 802.11 aa access categories are listed below in the table 2.3.

Table 2. 3: IEEE 802.11aa Access Category and User Priority

User Priority	Priority	Access Category	Transmit Queue	Designation
1	Lowest	AC_BK	BK	Background
2	⇓	AC_BK	BK	Background
0		AC_BE	BE	Best Effort
3		AC_BE	BE	Best Effort
4		A_VI	VI	Video Alternet
5		AC_VI	VI	Video
6		AC_VO	VO	Voice
7	Heighest	A_VO	VO	Voice Alternet

The alternate video (A_VI) and alternate voice (A_VO) transmit queues share the same EDCAF as VI and VO transmit queues, respectively, as shown in Figure above. When dot11Alternate EDCA Activated is true, a scheduling function above the VO EDCAF selects from the primary or alternate transmit queue to be the next to be passed to the VO EDCAF from the queue with the higher UP are selected with a higher probability than from the queue with the lower UP. When dot11AlternateEDCAActivated is true, a scheduling function above the VI EDCAF selects from the primary or alternate transmit queue to be the next to be passed to the VI EDCAF so that the traffic from the queue with the higher UP are selected with a higher probability than from the queue with the lower UP. The default algorithm to select a traffic from either the A_VI or VI queue or A_VO or VO queue.

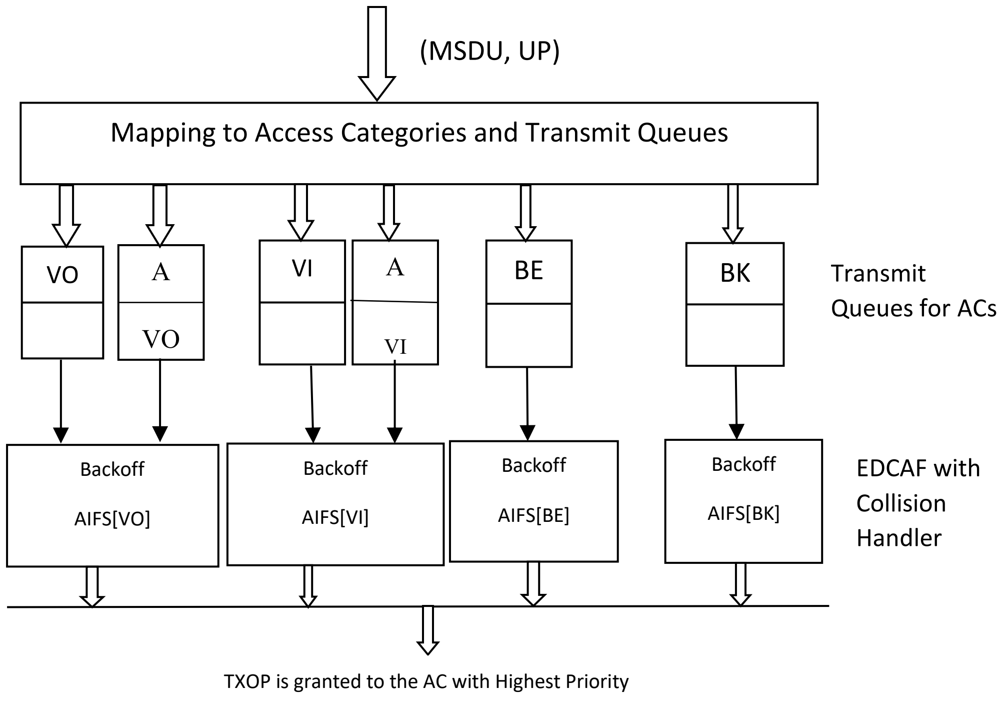
(MSDU, UP)
Mapping to Access Categories and Transmit Queues
VO
A
VO
VI
A
VI
BE
BK
Transmit Queues for ACs
Backoff
AIFS[VO]
Backoff
AIFS[VI]
Backoff
AIFS[BE]
Backoff
AIFS[BK]
EDCAF with Collision Handler
TXOP is granted to the AC with Highest Priority

Fig 2.11: IEEE802.11aa Queueing mechanism

2.3.1.2 IEEE 802.1Q Intra access Category priority

Intra Access Category priority selection done by based on IEEE 802.1Q scheduling [21]. This scheduling done by either of the following technique stated in 2.3.1.2.1 or 2.3.1.2.2.

2.3.1.2.1 Strict Priority Algorithm (SPA)

SPA is the default algorithm and gives absolute priority to the high priority queue, i.e., if there is at least one frame in the high priority queue it will be transmitted first. From [22] we can draw the SPA consists of following steps:

Strict Priority Algorithm

```
1   Loop
2   For i=NumOfTrafficClasses; i>=1; i++ do
3   If Queue(i)!= empty then
4   Frame = Queue(i).pop()
5   SendFrame(i) = SendFrame(frame)
```

From the above algorithm we can see that this is a simple algorithm it sends the data of which queues data is available in the priority basis and after finishing the transmission the data from that particular queue it transform to the next lower priority queue to check and transmits if found.

2.3.1.2.2 Credit Based Shaper Algorithm (CBSA)

This scheduler is configured so that frames from the primary queues are selected with a higher probability than frames from the alternate queues. A frame is queued at a time when the credit value is zero, and there is no conflicting traffic (there is no higher priority traffic awaiting transmission, and there is no frame being transmitted on the Port). The frame is immediately selected for transmission, and credit decreases at the rate of sendSlope as the transmission proceeds. Once the frame transmission is complete, credit increases back to zero at the rate of idleSlope, at which point, a further frame can be selected for transmission. The calculation of credit is based on the following two external parameters: portTransmitRate – the transmission rate, in bits per second, supported by the underlying MAC service, and idleSlope – the rate of change of credit, in bits per second, when the value of credit increases [23]. The maximum portion of portTransmitRate available for the alternative traffic is given as:

$$IdleSlope/PortTransmitRate. \quad (2.6)$$

Another internal parameter, sendSlope, determines the rate of credit change, in bits per second, when the value of credit decreases:

$$SendSlope = IdleSlope - PortTransmitRate. \quad (2.7)$$

The value of credit is increased with a rate of idleSlope in two cases:

(i) During the transmission of a frame from the high priority queue and

(ii) When there is no transmission while credit is negative.

Conversely, credit is decreased with a rate of sendSlope during the transmission of a frame from the alternate queue. Additionally, if credit is positive and there is no frame in the alternate queue then credit is reset to zero.

2.4 IEEE 802.11ax

As the users expectations increasing highly and the WLAN enabled devices is grown so rapidly, it is much needed for a technology to change as to cope up with this environment. To fulfil these demands IEEE working group on future wireless network is aimed to release a new version of wireless technology is IEEE 802.11 ax which will meet up the trends. Already, some new features proposed for IEEE 802.11AX PHY and MAC [24-26].

2.4.1 IEEE 802.11ax PHY

Target Wake Time (TWT): It is used for stations to sleep, save power and wake up at scheduled times. Thus an AP can schedule to minimize contention among stations. This can also lead to a load balancing technique to ease congestion.

BSS Color: In a dense network, a neighboring AP can cause interference to the other AP. STAs on the overlapping areas will back off excessively. Thus to alleviate this problem a feature is used called BSS Color. This helps stations to identify if transmission is from another network and thereby take the right action.

Transmit Beamforming: This is another great feature where an AP can use a number of transmit antennas to land a local maximum signal on a receiver's antennas. It improves data-rate and efficiently extends the range of service.

Modulation Scheme: In IEEE 802.11 ax higher order up to 1024-QAM will use which will increases peak data-rates under good conditions (high SNR).

2.4.2 IEEE 802.11ax MAC

Orthogonal Frequency Division Multiple Access (OFDMA): IEEE 802.11ax MAC adopts OFDMA which will allow multiple users to share the bandwidth at the same time which will speed up the transmission really. OFDMA utilizes bandwidth efficiently by allocating subcarriers to users based on their needs. Users also be experiencing less latency with OFDMA. With this technique, multiple users with varying bandwidth needs can be scheduled at the same time.

Uplink multiuser multiple input multiple output (MU-MIMO): This amendment will use uplink MU-MIMO technique which will schedule multiple users at a time.

Enhanced CSMA/CA (CSMA/ECA): A deterministic backoff is used after successful transmission, and backoff stage is not reset after service. The backoff stage is reset only when the STA decides to leave the contention. Moreover, the number of successfully transmitted packets is chosen as the function of the backoff stage to fairly deal with sharing of resources.

Spatial Reuse: The objective of SR operation is to improve the system-level performance, the utilization of medium resources, and power saving in dense deployment scenarios by early identification of signals from OBSSs and interference management. The improved system-level performance by HE spatial reuse is achieved by the enhanced channel access.

Random Access Protocol: According to different traffic and service requirements, any random access protocol can be used for random access in IEEE 802.11ax WLAN. However, instead of contending the medium in time domain, multiple HE STAs should be able to contend the medium in frequency domain to solve the bandwidth wastage due to collision(s), if multiuser PHY is adopted.

2.5 IEEE 802.11 amendments

Below we have listed some popular amendments of IEEE 802.11 with short description.

Table: IEEE 802.11 amendments

Name	Release	Features
802.11	1997	Original, data rate 2 mbps, 2.4 GHz stadard
802.11a	1999	54 mbps, 5 GHz standard
802.11b	1999	Extension of 802.11, 2.4 GHz to support 5.5 and 11 mbps
802.11g	2003	54 mbps , 2.4 GHz standard (extension of 802.11b)
802.11h	2003	5 GHz, Dynamic Channel /Frequency Selection (DCS/DFS) and Transmit Power Control (TPC).
802.11i	2004	Enhanced security, replaced conventional WEP.

802.11e	2005	Enhanced the 802.11 MAC protocol to support for QoS, included packet bursting.
802.11r	2008	Seamless, fast and secure handoff/roaming from one BSS to another.
802.11n	2009	Improves standard by MIMO in both 2.4 and 5 GHz bands to maximize data rate to 600 Mbps, improves security.
802.11i	2004	Enhanced security, replaced conventional WEP.
802.11p	2010	WAVE model - Wireless Access for the Vehicular Environment (such as ambulances and passenger cars etc.).
802.11s	2011	Wireless mesh networking topologies extension.
802.11u	2011	Interworking with other types (non 802 family) of networks (e.g., cellular).
802.11aa	2011	Robust Audio Video Streaming
802.11ac	2013	Extends 802.11n in 5 GHz band aims to offer higher throughput, higher density modulation, and additional MIMO streams to give a theoretical throughput of 7 Gbps in bands < 6 GHz.
802.11ah	2017	Wi-Fi HaLow, 900 MHz, Extends range, lower energy consumption, allow group of stations or sensors that cooperate to share signals, implementing IoT.
802.11ax	2020 (expected)	Aim to modify both 802.11 PHY layer and 802.11 MAC protocol, especially for densely deployed environments, works on 2.4 and 5 GHz band.

2.6 Challenges of IEEE 802.11 in Dense Deployment

After a wide deployment of Wi-Fi network and devices, we are facing some fundamental technology challenges especially in dense environment. One main reason behind that now IEEE 802.11 family has multiple PHY options, but there is only one common MAC option, which is CSMA/CA. According to D. Deng et. al. [26] the root-causes include:

Severe collisions from channel contention: As the current WLANs adopted IFS and CW to effectively control the MAC operation, vulnerability under dense deployment arises as a common problem since the MAC parameters of its collision avoidance/resolution mechanism are far from the optimal setting. First, the fundamental access method of IEEE 802.11, CSMA/CA, might incur with a high collision probability in dense scenarios and thus degrade the channel utilization. This is because MAC selects a small initial value of CW size by a naive assumption of a low level of congestion in the system. Second, CSMA/CA might lead to the fairness problem because its collision resolution algorithm, binary exponential backoff (BEB) algorithm, always favors the last successfully transmitted station. Finally, frame loss is another key factor diminishing the performance since MAC does not take into account the occurrence of non-collision losses. Unfortunately, wireless links are noisy and highly unreliable where Path loss, channel noise, fading, and interference may cause significant transmission errors. If the sender is unable to distinguish the causes of frame loss, it is difficult to make the right decision. To detect erratic errors and frame loss, an intelligent and cognitive protocol is therefore needed to ensure the system performance demanded by applications.

Increased Interference from Neighbors: Multiple AP with high density deployment may result in an overlapping problem, which cause inter-AP interference. Request-to-Send (RTS)/Clear-to-Send (CTS) mechanism is proposed to solve hidden terminal problem and enhance transferring performance. Hence, Hidden terminal problem occurs when there is a station under an AP, while trying to detect whether the channel is busy, is not aware of the ongoing transmission of another station. However, performance of the system may decreases as well. Another way to improve performance in dense environment is to increase spatial reuse by adopting dynamic Clear Channel Assessment (CCA). Furthermore, sometimes energy detection is not reliable for CCA in dense deployment because CCA is based on the energy received from the transmitter, irrespective of the recipient.

Also, as the WiFi implementation is increasing day by day and the real time Audio Video traffic is increasing a lot than other traffic, the mapping of these traffic wants to access the channel it now a burning issue. It is directly effect the QoE of user. As, the user viewing experience is related with it, it is goning to be necessary that there need to create a new procedure for channel access for the heavy and time critical traffic.

2.7 Related Work

Our study on related work divided in following categories. These are:

2.7.1 Work Related to IEEE 802.11e EDCA optimization

In [27], authors proposed an scheme which fragments IP packets based on the priority of video packets and the characteristics of MPEG-2 Transport Stream (TS), where the original IP packet is fragmented into smaller IP packets containing fewer TS packets and prioritizes individual TS packets, allocated to an appropriate priority queue. In [28], authors uses a technique to measure cumulative mean squared error (CMSE) for slice loss and prioritize the slices within a Group of Pictures (GoP) to improve video quality. In [29], authors designed a QoE-aware scheduling (QoEAS) scheme which derived a packet importance index at APP layer and embed it into the ToS field of the IP header, which improves video quality in dense user environment. In [30], the objective of the proposed scheme is to protect the I-Frames from being dropped by discarding the B-Frames from the AC_VI queue during congestion. In [31] author used a QoE Predictor and a Queue Controller to ensure protecting I-Frames. It ensures the quality of the delivered video frames.

2.7.2 Work related to IEEE 802.11aa parameter optimization

In [32], author designed a mathematical model for higher throughput of video frames over the IEEE 802.11aa. Wireless Credit Based Shaper Algorithm to prioritize traffic in primary and alternate queue of IEEE 802.11aa. After the credit reaches zero the alternate frame transmissions take place immediately. However, when both queues are loaded, the credit changes only during the actual frame transmissions. Additionally, after each transmission an acknowledgment and backoff procedures take place, thus without the adjustment period, the alternate queue obtains higher throughput. In [33], authors set up a separate token bucket builder for each queue, so that each queue can use a token bucket scheme that can flexibly control the flow and speed of the queue. When the data traffic exceeds the threshold, it shapes traffic to improve video transmission.

In [34] authors proposed a Single-Queue Priority Scheduler (SQPS) to rearrange video frames according to their importance. In [35], they have designed an algorithm called MUMAM that placed multicast flows in primary queue and unicast to secondary queue and then given priority to primary queue frames. In [36], authors designed an algorithm called Robust Wireless Credit Based Shaper Algorithm (RWCBSA) which modifies Credit value according to time spent on data bits transmission without the influence of duration of transmission procedure. After frame lifetime expiration the new frame transmission is treated as retransmission, so the Credit remains unchanged. It sets a threshold limit to the retransmission of frames by the collision factor. In [37], authors assessed the performance of different Group Addressed Transmission Service (GATS) mechanism under different scenarios for IEEE802.11aa network for multicast video delivery.

2.7.3 Related work on IEEE 802.11AX EDCA parameter optimization

In [38], authors solves the issue that legacy STAs suffers to get channel access in 802.11ax network. The AP adjusts its own EDCA parameters and also determines two sets of EDCA parameters (e,g, AIFS, CW) for associated STAs. The first one is for legacy STAs and it is disseminated via the legacy EDCA parameter set information element. The second one is for 802.11ax STAs and it is disseminated via a new information element. In [39], considers when a frame need to retransmit after getting no acknowledgement from destination which kept at the head of the transmit queue and, thus, blocks any other frames leads to head-of-line (HOL) blocking problem. Authors designed a schema that uses a set of backup queues as well as a two-stage scheduling approach to handle failing unicasts. In [40], a Channel Bonding based QoS-aware OFDMA MAC protocol (CBQO) is proposed. In CBQO, several 20MHz channels are firstly bonded as a wide channel by channel bonding and then divided into a number of sub channels by OFDMA. Both the multi-user channel access and the multi-user data transmission are implemented in the CBQO protocol, which means that a number of stations could access the network (by sending Enhanced Request To Send (E-RTS) packets to the AP) and transmit data packets concurrently on different sub channels. Thus it provides high throughput and QoS guarantee for video traffic in the dense user scenario of the IEEE 802.11ax. In [41], authors designed a uplink scheduler for 802.11ax network based on Shortest Remaining Processing Time (SRPT) using OFDMA, which makes uplink transmission faster. In [42], authors proposed an approach for choosing reserved intervals parameters in case of real time video transmission in dense cloudified IEEE 802.11aa/ax networks. Using a centralized control of a swarm of APs allows allocating them non-overlapping transmission intervals. In [43], authors an introduced a distributed reservation mechanism by setting optimal CW values for Enhanced Collision Avoidance- Distributed Reservation termed ECA-DR, based on which stations can collaboratively achieve higher network performance.

2.8 Limitations

These enhancements tend to address particular issues or applications by improving, in most cases, one of the HCF functions. In general, designing a robust HCF solution that provides an integrated solution for all traffic classes is still a challenging task for future research. Although all proposed schemes in their current states improve the transmission of video traffic, there still some issues need to be addressed and investigated [44].

2.8 Summary

In this chapter we have workout the basic DCF mechanism of CSMA/CA for wireless medium. We also have discussed the HCF mechanism for IEEE 802.11e and the HCF mechanism for IEEE 802.11aa for robust AV transmission. We also have discussed about the upcoming IEEE 802.11ax MAC features. Then we discussed the various literature work related to the above subjects in different scenarios. We also have drawn the limitation at last.

Chapter 3

Proposed Additional Video Access Queue Based EDCA (AVAQ-EDCA)

3.1 Protocol Architecture

In our proposed AVAQ-EDCA mechanism we extend the EDCA model and propose seven ACs. We have used two AC for Voice, AC DVO is for the HD Voice traffic and AC VO for the other voice traffics. Also, we have used three AC for video. These are KVI, DVI and VI. AC KVI for videos has 4K or upper resolutions. AC DVI for HD/2K/3K videos. AC VI for videos has resolutions lower than 1K. Best Effort and Background Data will map to one access category for each. There are four EDCAF with its own CW value for 7 ACs. Each of the seven ACs has its own priority value.

3.2 Priority Assignment

We have redistributed the priority value for our proposed AVAQ-EDCA protocol which has drawn in the below table 3.1.

Table 3.1: Priority Assignment of Proposed protocol

User Priority	Priority	Access Category	Designation
1	Lowest	AC_BK	Background
2		AC_BK	Background
0		AC_BE	Best Effort
3		AC_ VI	Video (Low Res)
4		AC_DVI	Video (HD)
5		AC_KVI	Video(4K)
6		AC_VO	Voice
7	Highest	AC_DVO	Voice HD

From table 3.1 above we can see that we have set highest priority 7 to the HD Voice traffic which map to AC DVO. Also, other voice traffic to AC VO with a priority value of 6. We have created 3 access category for video traffic. We assigns priority value 5 to the Higher Resolutions Video 4K, 8K etc. and map to AC KVI , HD/2K resolutions video to AC DVI and assigns priority 4 , Low resolution (Less than 1K) video traffic assigns to AC VI with a priority value of 3. We assigns best effort traffic to AC BE with a priority value 0 and background traffics to AC BK with priority values 2 and 1.

3.3 AVAQ-EDCA Working Principle

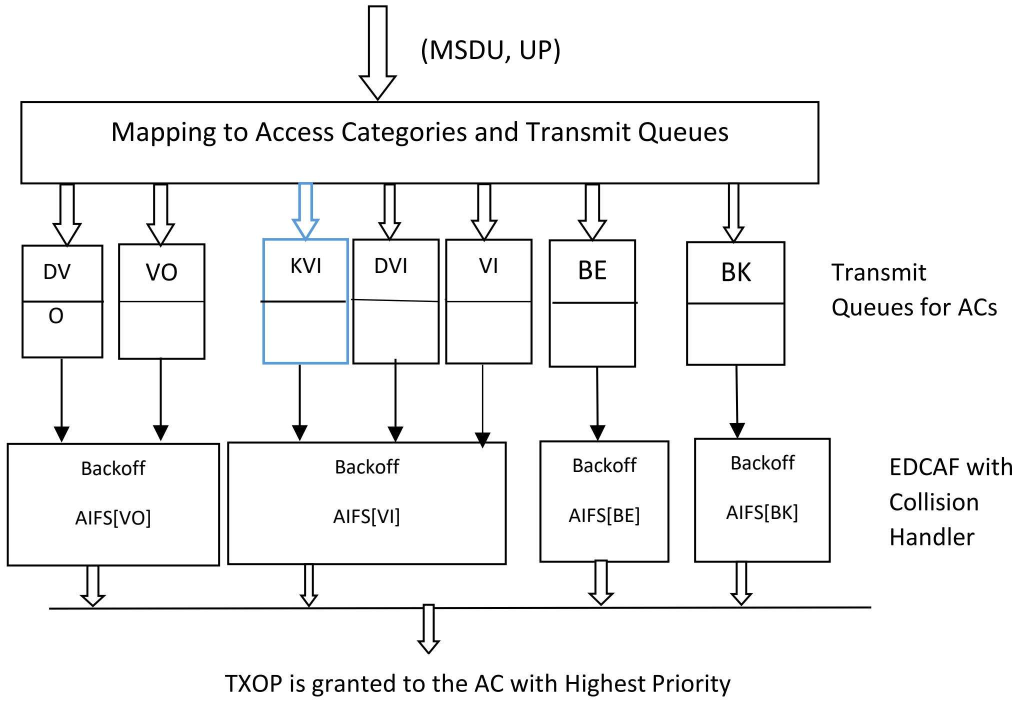

Figure 3.1: Proposed AVAQ-EDCA internal Mechanism

From figure we see that we have added a new access queue called KVI to the model IEEE 802.11 aa. This additional queue is intended for handling the HDR 4K or above resolutions video traffic if they can get a priories access to the channel. Others two queue DVI and VI will be used Full HD and HD or lower resolutions video traffic respectively. We have also change in the functionality and naming of the two queues for audio traffic. These two queues DVO and VO will be used to serve HD voice and regular voice respectively. We have implemented an algorithm to serve the additional queue for transmission. This algorithm shall be used to pick differentiated traffic based on their assigned priority and send to the channel. The EDCAF will control the transmission procedure and internal collision resolution. The higher priority AC will be selected for the transmission with the specified TXOP limit. We have used the TXOP limit value from IEEE 802.11-2016 [45] for the Voice traffic is 2.080 to 3.264 ms and for the video traffic is 3. 264 to 6.016 ms. The Background and Best effort traffic will transmit with the TXOP limit value of 0 ms that these can transmit only once at a time.

3.4 AVAQ-EDCA algorithm

Our proposed method developed based on the SPA algorithm discussed in 2.3.1.2.1.

Algorithm 1: Multi queue video transmission algorithm

```
    Let $\wp$k, $\wp$d, $\wp$i represents the Queue for AC_KVI, AC_DVI and AC_VI.
1.  Check $\wp$k, $\wp$d, $\wp$i for frames
2.  If $\wp$k $\neq 0$ then
3.      frame = $\wp$k. pop()
4.      transmit $\leftarrow$ frame($\wp$k)
5.          else if $\wp$d $\neq 0$ then
6.              frame = $\wp$d. pop()
7.              transmit $\leftarrow$ frame
8.                  else if $\wp$i $\neq 0$ then
9.                      frame = $\wp$i. pop()
10.                     transmit $\leftarrow$ frame
11.                 end if
12.         end if
13. end if
```

Algorithm 2: Multi queue Audio transmission algorithm

```
     Let ф d, ф o represents the Queue for AC_DVO, AC_VO
1.   Check ф d, ф o for frames
2.   If ф d $\neq 0$ then
3.       frame = ф d. pop()
4.       transmit ← frame
5.           else if ф o $\neq 0$ then
6.               frame = ф o. pop()
7.               transmit ← frame
12.          end if
13.  end if
```

3.5 Summary

In this chapter an additional video traffic serving queue based algorithm proposed to the addition of IEEE 802. 11 aa and IEEE 802. 11e EDCA model. Our model handles video traffic based on 3 different classes. It may lead us to the fair scheduling of the priories video traffic to the channel.

Chapter 4

Results and Performance Evolution

In this section, performance of the proposed AVAQ-EDCA protocol evaluated through simulation study. For performance analysis of the proposed AVAQ-EDCA protocol compared with IEEE 802.11e EDCA and one of the recent IEEE 802.11 aa based Robust Wireless Credit Based Shaper Algorithm (RWCBSA) by using ns3 simulator [46].

4.1 Simulation Environment

The ns-3 is a discrete-event network simulator and it is publicly available for research, development, and use. The ns-3 simulation supports both IP and non-IP based networks. It is a popular tools for simulating Wi-Fi networks.

We use this tool for simulating our proposed AVAQ-EDCA model. In our model we have used one AP and varying number of STAs. We use single channel scenario that varying number of STA's contending for accessing a single channel in IEEE 802.11 ax based AVAQ-EDCA model. The AP is placed in the middle of the STA's with fixed position throughout the simulation period and STA's movement controlled by the Random walk 2d or Random waypoint model. We have placed each STA 10 meter apart from each other and followed grid architecture to set up the scenario. We have designed few scenarios by varying number of QoS supported Stations (QSTAs). We simulate all the three model (EDCA, RWCBSA and proposed AVAQ-EDCA) with the scenario file with 4 STA and one AP. Then we have simulate all the three model (EDCA, RWCBSA and proposed AVAQ-EDCA) with the scenario file with 8 STA and one AP. After that we simulate all the three model (EDCA, RWCBSA and proposed AVAQ-EDCA) with the scenario file with 12 STA and one AP, 16 STA and one AP and 19 STA and one AP respectively. The scenario with 19 STA and one AP in ns3 is as follows:

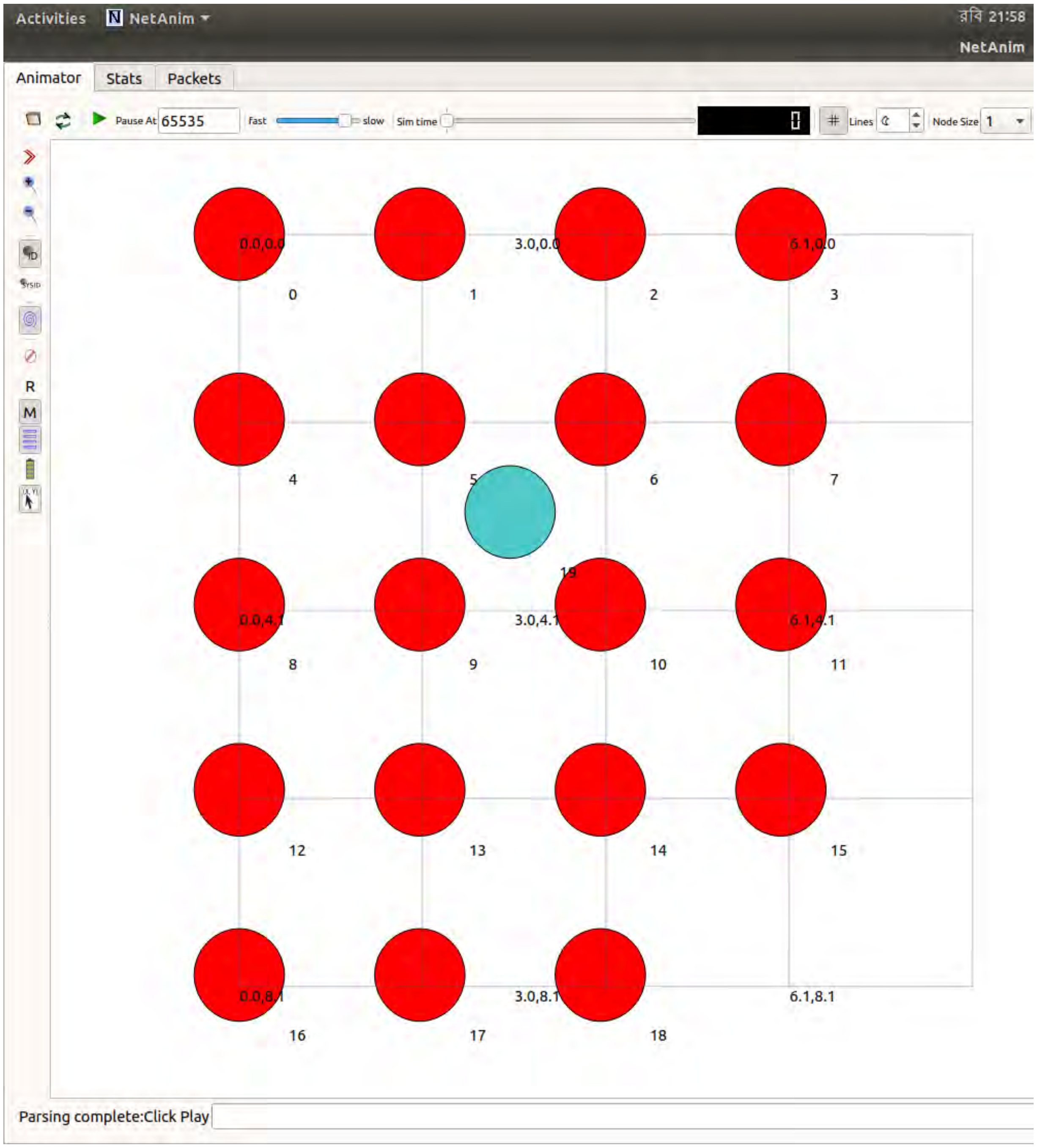

Figure 4.1: Simulation model in ns3 with proposed AVAQ-EDCA protocol.

Other parameters that we have used to simulate our protocol are as follows:

Table 4.1: Simulation Parameter

SI. No.	Parameter	Value
1	Data Packet Size	1500 Bytes
2	Max Stations	19
3	No. of Packets	10,000 – 100,000
4	Packets Per Second	0.0002
5	Simulation Time	10 – 20 seconds
6	Mobility	Random waypoint, Random walk2d
7	Distance (One STA to Other STA)	10 meter

We have edited the source file of the directory ns3.29/src/model/wifi/qos-utils.cc, qos-utils.h, regular-wifi-mac.cc, wifi-mac.cc, edca-paramer-set.cc, edca-parameter-set.h to add an additional queue to the existing model and redistribute priority value. Then, we have simulated three modles (Default EDCA, IEEE802.11 aa based RWCBSA and our proposed AVAQ-EDCA) with our scenario file scenario.cc (appendix a) with varying number of STAs. We use netanim of ns3 to get the mean delay, packet delivery ratio and throughput of the scenario.

4.2 Simulation Results

We have used gnuplot [47] to plot the results collected from netanim after running scenario file programs. This gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed. It was originally created to allow scientists and students to visualize mathematical functions and data interactively. We have used this utility to plot our collected results. We have run our scenario file to get this result in 3 different model (EDCA, RWCBSA and our proposed AVAQ-EDCA) to compare which performs the best. The code used for getting these plots are mentioned in the Appendix.

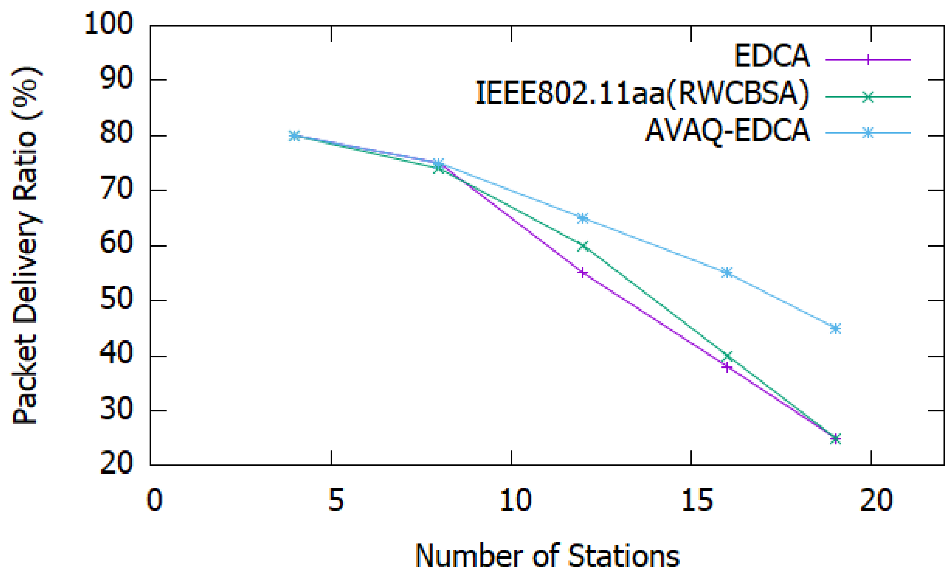

Figure 4.2: Packet Delivery Ratio vs No. of Stations (No. of Packets 10,000)

As we see from this figure 4.2 above that the IEEE 802. 11 aa based RWCBSA protocols PDR is good enough up to the 12 contending stations, but decreases when the number of station increases to 16 or more. We have achieved a higher packet delivery ratio than the other two model for our proposed AVAQ-EDCA protocol until the contending stations up to 19. We also have simulated by increasing the number of packets (from figure 4.3) in each STA up to 100,000 and achieved similar performance ratio regarding PDR.

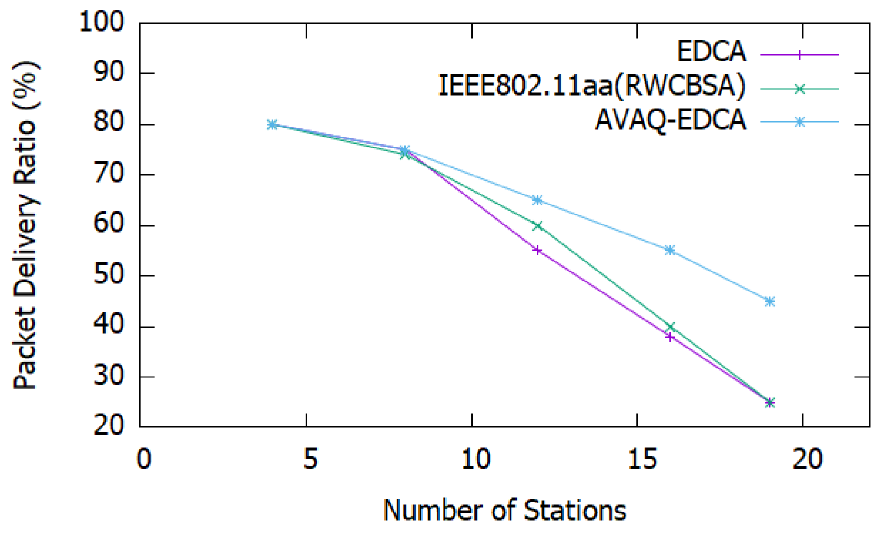

Figure 4.3: Packet Delivery Ratio vs No. of Stations (No. of Packets 100,000)

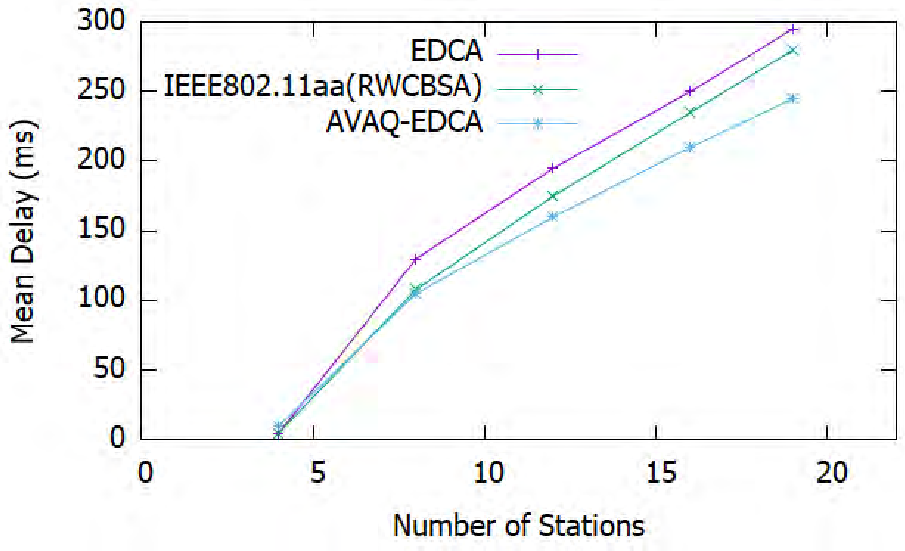

Figure 4.4: Delay vs No. of Stations (No. of Packets 10,000)

As we can see from this figure 4.4 above that the IEEE 802. 11 aa based RWCBSA protocols delay is good enough up to the 8 contending stations, but increasing when the number of station increasing to 12 or more. We have achieved a less mean delay for our proposed AVAQ-EDCA protocol when the contending stations up to 19. We also have simulated by increasing the number of packets (from figure 4.5) in each STA up to 100,000 and achieved similar performance ratio regarding delay.

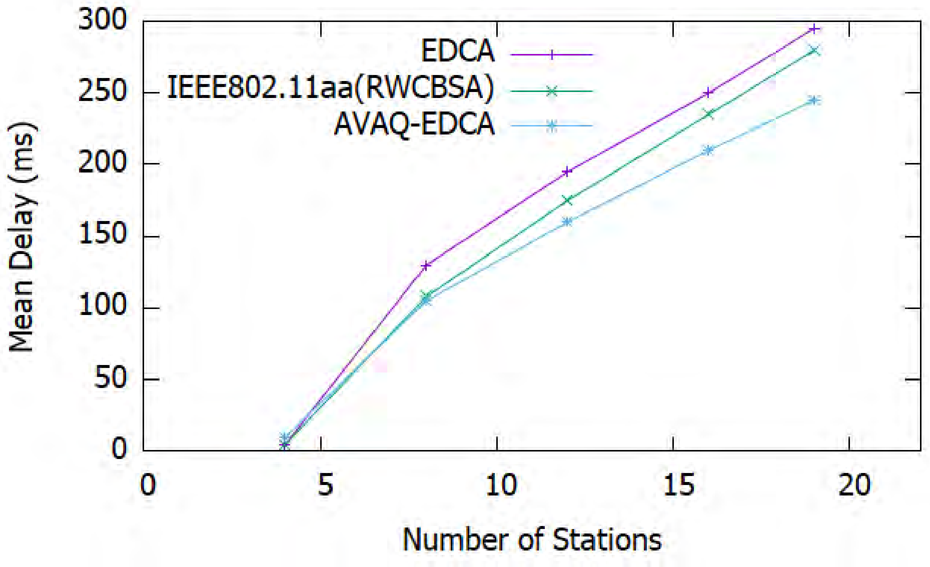

Figure 4.5: Delay vs No. of Stations (No. of Packets 100,000)

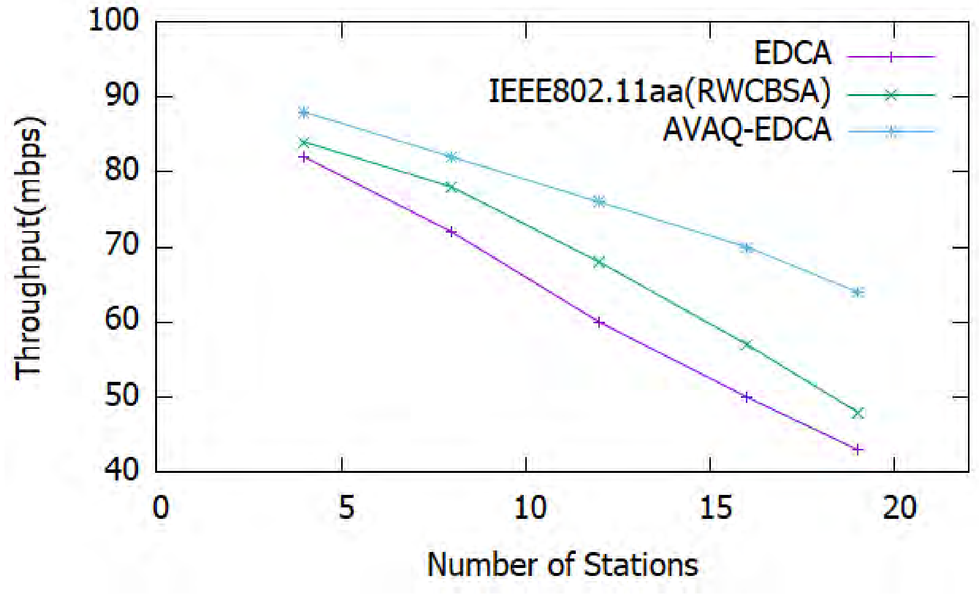

Figure 4.6: Throughput vs No. of Stations (No. of Packets 10,000)

As we see from this figure 4.6 above and 4.7 below that we have achieved higher throughput for proposed AVAQ-EDCA protocol than the IEEE 802. 11 aa based RWCBSA protocol and EDCA protocol. In both cases for number of packets 10,000 and 100,000.

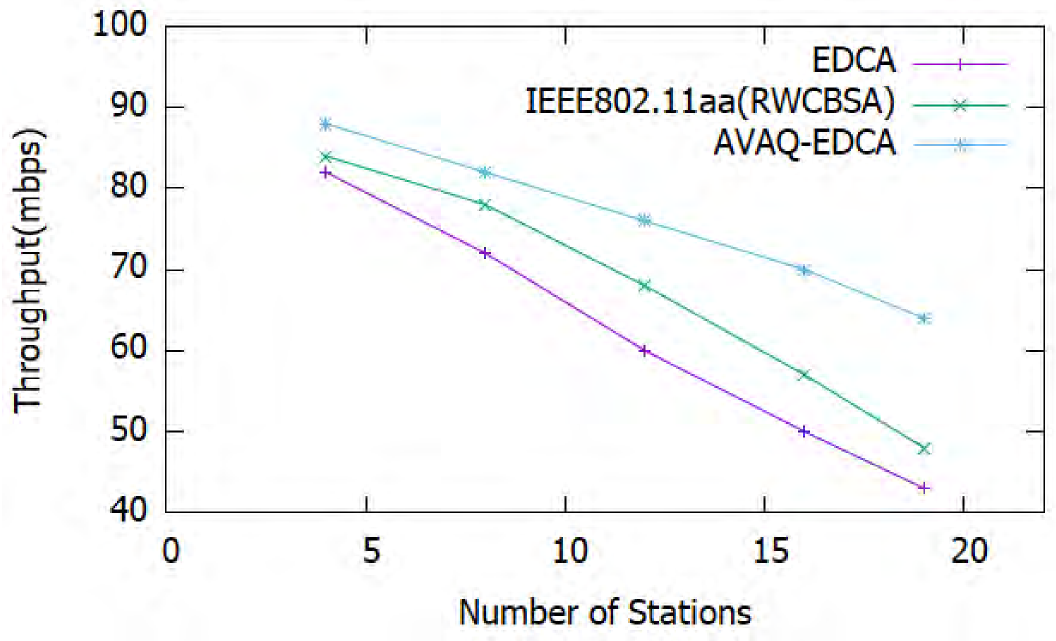

Figure 4.7: Throughput vs No. of Stations (No. of Packets 100,000)

4.3 Summary

We have conducted simulation to measure the efficiency of our proposed protocol. From simulation results we can see that our protocol outperforms other similar protocols. As a performance metrics we consider Mean delay, PDR and throughput.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

A lot of uplink scheduling protocol has been introduced in the last few years. Most of them considers the adjustment of contention window, different intra queue selection process, overhead management etc. But, most of them cannot consider the new approach for mitigating the need of various types of video traffic handling. In this research work, we proposed a model for handling the differential types of video traffic which is the most demanding for future networks. This thesis presents a new additional queue for handling HDR 4K 8K video traffic, giving them priority to the channel access. This research also improves the packet delivery ratio to the network also minimizes packet latency and overall improves the throughput. We prove that our proposed method performs well for a heavy traffic network. Also it can handle fairly the bigger packets than the other model.

NS3 simulator is used to compare the results of the proposed scheme with IEEE 802.11e EDCA and one of the recently proposed IEEE 802.11aa based RWCBSA. All results show that the network performance of the proposed scheme is better than previously proposed clustering techniques. This improvement is achieved using balanced selection of traffic in intra-queue periodization technique.

5.2 Future Work

The proposed protocol that generally the video traffic selection intra-queue handing for video traffic in the heavy load of the network. A further study will be useful to discover the mathematical modeling of the proposed AVAQ-EDCA protocol. So, we can say, the UHD video traffic selection procedure for the proposed queue can be added to the proposed protocol to make the approach more efficient and reliable.

References

[1] "Wireless LAN medium access control (mac) and physical layer (phy) specifications," *IEEE Std. 802.11-1997*, pp. 1 – 445, 1997.

[2] "Medium Access Control (MAC) Quality of Service Enhancements", *IEEE Std. 802.11e-2005*, pp. 1-212, 2005.

[3] "MAC Enhancements for Robust Audio Video Streaming", IEEE Std. 802.11aa-2012, pp. 1-162, 2012.

[4] "Enhancements for High Efficiency WLAN", *IEEE P802.11ax/D3*, pp. 1-682, 2018.

[5] Bellalta, B., "IEEE 802.11ax: High-efficiency WLANS," *IEEE Wireless Commun. Mag.,* Vol. 23, no.1, pp. 38–46, 2016.

[6] Khorov, E., Kiryanov, A., Lyakhov, A., & Bianchi, G., "A Tutorial on IEEE 802.11ax High Efficiency WLANs"*, IEEE Communications Surveys & Tutorials, 2018.*

[7] Cisco Inc., "*Cisco Visual Networking Index: Forecast and Methodology, 2014 – 2019*", 2015.

[8] Cisco Inc., "*Cisco Visual Networking Index: Forecast and Trends, 2017 –* 2022", 2018.

[9] Cisco Inc., https://www.cisco.com/en/US/products/hw/wireless/ps430/prod_technical_reference09186a0080144498.html [Last access on 26 February 2019].

[10] Panchadcharam, S., Ni, Q., Taylor, G., "Evaluation of QoS Parameters for Real-time Rich Media Applications Using a Test-bed Implementation", *10th IEEE International Conference on Computer and Information Technology),* pp. 343 – 348, 2010.

[11] Kosek-Szott, K., “What's new for QoS in IEEE 802.11?”, *IEEE Journals & Magazines*, Vol. 27, Issue 6 , pp. 95-104, 2013.

[12] Wu, H., Pan, Y., “Medium Access Control in Wireless Networks” *Nova Science Publishers*, Inc., 2008.

[13] Mangold, S., Choi, S., May, P ., Klein, O., Hiertz, G., Stibor, L., “IEEE 802.11e Wireless LAN for Quality of Service”, *Proc. European Wireless 2*, pp. 32-39, 2002.

[14] Chen, D., Garg, S., Kappes, M., Trivedi, K.S., “Supporting VBR VoIP traffic with IEEE802.11 WLAN in PCF mode”, *In Proceedings of OPNET Work,* 2002.

[15] Grilo, A., Nunes, M., “Performance evaluation of IEEE 802.11e”, *in Proc. Conf. IEEE Persona Indoor Mobile Radio Communications (PIMRC)*, pp. 511–517, 2002.

[16] Cicconetti, C., Lenzini, L., , Mingozzi, E., Stea, G., “Design and performance analysis of the Real-Time HCCA scheduler for IEEE 802.11e WLANs”, *Computer Networks*, Vol. 51, Issue 9, pp. 2311–2325, 2007.

[17] Hui, J., Devetsikiotis, M., “A Unified Model for the Performance Analysis of IEEE 802.11e EDCA”, *IEEE TRANSACTIONS ON COMMUNICATIONS,* Vol. 53, No. 9, *2005.*

[18] Huang, C. L., Liao, W., “Throughput and Delay Performance of IEEE 802.11e Enhanced Distributed Channel Access (EDCA) Under Saturation Condition”, *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS,* Vol. 6, No. 1, 2007.

[19] Rashwand, S., “IEEE 802.11e EDCA Under Bursty Traffic—How Much TXOP Can Improve Performance”, *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, Vol. 60, No. 3, 2011.

[20] Kong, Z.N., “Performance Analysis of IEEE 802.11e Contention-Based Channel Access”, *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, Vol. 22, No. 10, 2004.

[21] “Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks”, *IEEE Std. 802.1Q-2011*, pp. 1-1365, 2011.

[22] Zawia, H.I., Hassan, R., Dhanil, D.P., “A Survey of Medium Access Mechanisms for Providing Robust Audio Video Streaming in IEEE 802.11aa Standard”, *IEEE Access*, vol. 6, pp. 27 690–27 705, 2018.

[23] Kosek-Szott, K., Natkaniec, M., Prasnal, L., “A Novel IEEE 802.11aa Intra-AC Prioritization Method for Video Transmissions”, *IEEE Global Communications Conference*, pp. 1158 – 1163, 2014.

[24] Ali, R., Kim, S. W., Kim, B. S., and Park, Y., “Design of MAC Layer Resource Allocation Schemes for IEEE 802.11ax: Future Directions”, *IETE Technical Review*, vol. 35, Issue 1, pp. 28-52, 2016.

[25] Deng, D., Lin, Y., Yang, X., Zhu, J. et. al, “IEEE 802.11ax: Highly Efficient WLANs for Intelligent Information Infrastructure”, *IEEE Communications Magazine,* pp. 52-59*,* 2017.

[26] Deng, D. J., Lien, S. Y., Lee, J., and Chen, K. C., “On quality-of-service provisioning in IEEE 802.11ax WLANs,” *IEEE Access*, vol. 4, pp. 6086–6104, 2016.

[27] Shin, B., Abdullayev, J., Lee, D., “An Efficient MAC Layer Packet Fragmentation Scheme with Priority Queuing for Real-Time Video Streaming”, *IEEE 41st Conference on Local Computer Networks (LCN)*, pp. 69 – 77, 2016.

[28] Paluri, S., Kambhatla, K.K.R., Bailey, B.A. et al.” A low complexity model for predicting slice loss distortion for prioritizing H.264/AVC video”, *Multimed Tools Appl*, Volume 75, Issue 2, pp. 961–985, 2016.

[29] Li, M., Tan, P.H, Sun, S., Chew, Y.H., QoE-Aware Scheduling for Video Streaming in 802.11n/ac-Based High User Density Networks, *IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pp 1-5, 2016.

[30] Khambari, N., Ghita, B., Lancaster, D., Sun, L., “QoE Enhancements in IEEE 802.11 e EDCA for Video Transmission through Selective Queueing”, *INC*, pp- 85-90, 2016.

[31] Khambari, N., Ghita, B., Sun, L., “QoE-driven video enhancements in wireless networks through predictive packet drops”, *IEEE 13th International Conference on Wireless and Mobile Computing*, *Networking and Communications (WiMob)*, 2017.

[32] Kosek-Szott, K., “A Throughput Model of IEEE 802.11aa Intra-Access Category Prioritization”, *Wireless Personal Communications*, Vol. 71, Issue 2, pp- 1075–1083, 2013.

[33] Zhang, Y., Liu, S., Zhang, R., Wei, W., Song, H., Li, W., Fan, X., "A New Multi-Service Token Bucket-Shaping Scheme Based on 802.11e", *International Conference on Identification, Information, and Knowledge in the Internet of Things (IIKI)*, pp. 197 – 200, 2015.

[34] Adeleke, J.A., Dlodlo, M.E., Onime, C.E., "Video Traffic Prioritization in WLANs Using Single Queue Priority Scheduler", *17th UKSim-AMSS International Conference on Modelling and Simulation*, pp. 557 – 562, 2015.

[35] Charfi, E., Chaari, L., Hlima, S.B., Kammoun, E., "Multi-user access mechanism with intra-access categories differentiation for IEEE 802.11ac wireless local area networks", *Telecommunications Systems journal*, Volume 64 Issue 3, pp. 479-494, 2017.

[36] Prasnal, L., Natkaniec, M., "Robust Traffic Differentiation with Intra-AC Prioritization in Multirate IEEE 802.11aa Networks", *IEEE Symposium on Computers and Communications (ISCC)*, pp. 1278-1283, 2017.

[37] Gringoli, F., Serrano, P., Facchi, N., Azcorra, A., "Experimental QoE evaluation of multicast video delivery over IEEE 802.11aa WLANs", *IEEE Transactions on Mobile Computing*, pp.1-1, 2018.

[38] Khorov, E., Loginov, V., and Lyakhov, A., "Several EDCA parameter sets for improving channel access in IEEE 802.11ax networks," In *International Symposium on Wireless Communication Systems (ISWCS)*, pp. 419–423, 2016.

[39] Pannu, G. S., Klingler, F., Sommer, C., "QQDCA: Adapting IEEE 802.11 EDCA for Unicast Transmissions at High Topology Dynamics", *IEEE Vehicular Networking Conference (VNC),* pp. 295 – 302, 2017.

[40] Zhou, H., Li, B., Yan, Z., and Yang, M., "A Channel Bonding Based QoS-Aware OFDMA MAC Protocol for the Next Generation WLAN", *Mobile Networks and Applications*, Vol. 22, Issue 1, pp 19–29, 2017.

[41] Bankov, D., Didenko, A., Khorov, E., and Lyakhov, A., "OFDMA Uplink Scheduling in IEEE 802.11ax Networks", *IEEE International Conference on Communications (ICC)*, pp. 1-6, 2018.

[42] Khorov, E. Ivanov, A., Lyakhov, , A., Akyildiz, I. F., “Cloud Control to Optimize Real-Time Video Transmission in Dense IEEE 802.11aa/ax Networks”*, IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS),* pp. 193 – 201, 2018.

[43] Salehi, S., Li, L., Shen, C., Cimini, L, Graybeal, G., “Traffic Differentiation in Dense WLANs with CSMA/ECA-DR MAC Protocol”, *IEEE VTC Fall conference,* 2018.

[44] Al-Maqri, M., Alrashah, M., Othman, M., “Review on QoS Provisioning Approaches for Supporting Video Traffic in IEEE802.11e: Challenges and Issues”, *IEEE Access*, vol. 6, pp. 55202-55219, 2018.

[45] “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, *IEEE Std 802.11-2016*, pp. 1 – 3534, 2016.

[46] Network Simulator NS3, https://www.nsnam.org/ [Last access on 15 March 2019].

[47] Gnuplot, http://www.gnuplot.info/ [Last access on 15 March 2019].

Appendix A

Simulation Source Code

Qos-utils.cc

```
AcIndex
QosUtilsMapTidToAc (uint8_t tid)
{
  NS_ASSERT_MSG (tid < 8, "Tid " << +tid << " out of range");
  switch (tid)
    {
    case 0:
    return AC_BE;
    break;
    case 1:
    case 2:
    return AC_BK;
    break;
    case 3:
    return AC_VI;
    break;
    case 4:
    return AC_DVI;
    break;
    case 5:
```

```
    return AC_KVI;
    break;
    case 6:
    return AC_VO;
    break;
    case 7:
    return AC_DVO;
    break;
    }
  return AC_UNDEF;
}

}

}
```

Qos-utils.h

```
enum AcIndex
{
  AC_BE = 0,
  AC_BK = 1,
  AC_VI = 2,
  AC_DVI = 3,
  AC_KVI = 4,
  AC_VO = 5,
  AC_DVO = 6,
  AC_BE_NQOS = 7,
  AC_UNDEF
};
}
```

edca-parameter-set.cc

```
namespace ns3 {
EdcaParameterSet::EdcaParameterSet ()
  : m_qosInfo (0),
    m_reserved (0),
    m_acBE (0),
    m_acBK (0),
    m_acVI (0),
    m_acDVI (0),
    m_acKVI (0),
    m_acVO (0),
    m_acDVO (0),
    m_qosSupported (0)
}

uint8_t
EdcaParameterSet::IsQosSupported (void) const
{
  return ((m_acBE != 0) || (m_acBK != 0) ||(m_acVI != 0)||(m_acDVI != 0) || (m_acKVI != 0) ||
(m_acVO != 0) ||(m_acDVO != 0));
}
```

```
void
EdcaParameterSet::SetDviAifsn (uint8_t aifsn)
{
  m_acDVI |= (aifsn & 0x0f);
}

void
EdcaParameterSet::SetDviAci (uint8_t aci)
{
  m_acDVI |= (aci & 0x03) << 5;
}

void
EdcaParameterSet::SetDviCWmin (uint32_t cwMin)
{
  uint8_t ECWmin = static_cast<uint8_t> (log2 (cwMin + 1));
  m_acDVI |= (ECWmin & 0x0f) << 8;
}

void
EdcaParameterSet::SetDviCWmax (uint32_t cwMax)
{
  uint8_t ECWmax = static_cast<uint8_t> (log2 (cwMax + 1));
  m_acDVI |= (ECWmax & 0x0f) << 12;
}

void
EdcaParameterSet::SetDviTxopLimit (uint16_t txop)
```

```
{
  m_acDVI |= txop << 16;
}

void
EdcaParameterSet::SetKviAifsn (uint8_t aifsn)
{
  m_acKVI |= (aifsn & 0x0f);
}

void
EdcaParameterSet::SetKviAci (uint8_t aci)
{
  m_acKVI |= (aci & 0x03) << 5;
}

void
EdcaParameterSet::SetKviCWmin (uint32_t cwMin)
{
  uint8_t ECWmin = static_cast<uint8_t> (log2 (cwMin + 1));
  m_acKVI |= (ECWmin & 0x0f) << 8;
}

void
EdcaParameterSet::SetKviCWmax (uint32_t cwMax)
{
  uint8_t ECWmax = static_cast<uint8_t> (log2 (cwMax + 1));
  m_acKVI |= (ECWmax & 0x0f) << 12;
}

void
EdcaParameterSet::SetKviTxopLimit (uint16_t txop)
{
  m_acKVI |= txop << 16;
}



void
EdcaParameterSet::SetDvoAifsn (uint8_t aifsn)
{
  m_acDVO |= (aifsn & 0x0f);
}

void
EdcaParameterSet::SetDvoAci (uint8_t aci)
{
  m_acDVO |= (aci & 0x03) << 5;
}

void
EdcaParameterSet::SetDvoCWmin (uint32_t cwMin)
{
  uint8_t ECWmin = static_cast<uint8_t> (log2 (cwMin + 1));
  m_acDVO |= (ECWmin & 0x0f) << 8;
}
```

```
void
EdcaParameterSet::SetDvoCWmax (uint32_t cwMax)
{
  uint8_t ECWmax = static_cast<uint8_t> (log2 (cwMax + 1));
  m_acDVO |= (ECWmax & 0x0f) << 12;
}

void
EdcaParameterSet::SetDvoTxopLimit (uint16_t txop)
{
  m_acDVO |= txop << 16;
}

uint8_t
EdcaParameterSet::GetDviAifsn (void) const
{
  return (m_acDVI & 0x0f);
}

uint32_t
EdcaParameterSet::GetDviCWmin (void) const
{
  uint8_t ECWmin = ((m_acDVI >> 8) & 0x0f);
  return static_cast<uint32_t> (exp2 (ECWmin) - 1);
}

uint32_t
EdcaParameterSet::GetDviCWmax (void) const
```

```
{
  uint8_t ECWmax = ((m_acDVI >> 12) & 0x0f);
  return static_cast<uint32_t> (exp2 (ECWmax) - 1);
}

uint16_t
EdcaParameterSet::GetDviTxopLimit (void) const
{
  return (m_acDVI >> 16);
}

uint8_t
EdcaParameterSet::GetKviAifsn (void) const
{
  return (m_acKVI & 0x0f);
}

uint32_t
EdcaParameterSet::GetKviCWmin (void) const
{
  uint8_t ECWmin = ((m_acKVI >> 8) & 0x0f);
  return static_cast<uint32_t> (exp2 (ECWmin) - 1);
}

uint32_t
EdcaParameterSet::GetKviCWmax (void) const
{
  uint8_t ECWmax = ((m_acKVI >> 12) & 0x0f);
  return static_cast<uint32_t> (exp2 (ECWmax) - 1);
}

uint16_t
EdcaParameterSet::GetKviTxopLimit (void) const
{
  return (m_acKVI >> 16);
}



uint8_t
EdcaParameterSet::GetDvoAifsn (void) const
{
  return (m_acDVO & 0x0f);
}

uint32_t
EdcaParameterSet::GetDvoCWmin (void) const
{
  uint8_t ECWmin = ((m_acDVO >> 8) & 0x0f);
  return static_cast<uint32_t> (exp2 (ECWmin) - 1);
}

uint32_t
EdcaParameterSet::GetDvoCWmax (void) const
{
  uint8_t ECWmax = ((m_acDVO >> 12) & 0x0f);
  return static_cast<uint32_t> (exp2 (ECWmax) - 1);
```

```
}

uint16_t
EdcaParameterSet::GetDvoTxopLimit (void) const
{
  return (m_acDVO >> 16);
}

void
EdcaParameterSet::SerializeInformationField (Buffer::Iterator start) const
{
  if (m_qosSupported)
    {
      start.WriteU8 (GetQosInfo ());
      start.WriteU8 (m_reserved);
      start.WriteU32 (m_acBE);
      start.WriteU32 (m_acBK);
      start.WriteU32 (m_acVI);
      start.WriteU32 (m_acDVI);
      start.WriteU32 (m_acKVI);
      start.WriteU32 (m_acVO);
      start.WriteU32 (m_acDVO);
    }
}

uint8_t
EdcaParameterSet::DeserializeInformationField (Buffer::Iterator start, uint8_t length)
{
  Buffer::Iterator i = start;
  m_qosInfo = i.ReadU8 ();
  m_reserved = i.ReadU8 ();
  m_acBE = i.ReadU32 ();
  m_acBK = i.ReadU32 ();
  m_acVI = i.ReadU32 ();
  m_acDVI = i.ReadU32 ();
  m_acKVI = i.ReadU32 ();
  m_acVO = i.ReadU32 ();
  m_acDVO = i.ReadU32 ();
  return length;
}

}
```

edca-parameter-set.h

```
#ifndef EDCA_PARAMETER_SET_H
#define EDCA_PARAMETER_SET_H

#include "wifi-information-element.h"

namespace ns3 {

class EdcaParameterSet : public WifiInformationElement
{
public:
  EdcaParameterSet ();
```

```
void SetDviAifsn (uint8_t aifsn);
void SetDviAci (uint8_t aci);
void SetDviCWmin (uint32_t cwMin);
void SetDviCWmax (uint32_t cwMax);
void SetDviTxopLimit (uint16_t txop);
void SetKviAifsn (uint8_t aifsn);
void SetKviAci (uint8_t aci);
void SetKviCWmin (uint32_t cwMin);
void SetKviCWmax (uint32_t cwMax);
void SetKviTxopLimit (uint16_t txop);
void SetDvoAifsn (uint8_t aifsn);
void SetDvoAci (uint8_t aci);
void SetDvoCWmin (uint32_t cwMin);
void SetDvoCWmax (uint32_t cwMax);
void SetDvoTxopLimit (uint16_t txop);
uint8_t GetDviAifsn (void) const;
uint32_t GetDviCWmin (void) const;
uint32_t GetDviCWmax (void) const;
uint16_t GetDviTxopLimit (void) const;
uint8_t GetKviAifsn (void) const;
uint32_t GetKviCWmin (void) const;
uint32_t GetKviCWmax (void) const;
uint16_t GetKviTxopLimit (void) const;
uint8_t GetDvoAifsn (void) const;
uint32_t GetDvoCWmin (void) const;
uint32_t GetDvoCWmax (void) const;
uint16_t GetDvoTxopLimit (void) const;
```

```
WifiInformationElementId ElementId () const;
uint8_t GetInformationFieldSize () const;
void SerializeInformationField (Buffer::Iterator start) const;
uint8_t DeserializeInformationField (Buffer::Iterator start, uint8_t length);

  Buffer::Iterator Serialize (Buffer::Iterator start) const;
  uint16_t GetSerializedSize () const;



private:
  uint8_t m_qosInfo;
  uint8_t m_reserved;
  uint32_t m_acBE;
  uint32_t m_acBK;
  uint32_t m_acVI;
  uint32_t m_acDVI;
  uint32_t m_acKVI;
  uint32_t m_acVO;
  uint32_t m_acDVO;
};

}
```

regular-wifi-mac.cc

```
  SetupEdcaQueue (AC_DVO);
  SetupEdcaQueue (AC_VO);
  SetupEdcaQueue (AC_KVI);
  SetupEdcaQueue (AC_DVI);
  SetupEdcaQueue (AC_VI);
  SetupEdcaQueue (AC_BE);
  SetupEdcaQueue (AC_BK);
}



HeCapabilities
RegularWifiMac::GetHeCapabilities (void) const
{
  NS_LOG_FUNCTION (this);

  HeCapabilities capabilities;
 double maxAmpduLengthExponent1 = std::max (std::ceil ((std::log (std::max (std::max
(m_beMaxAmpduSize, m_bkMaxAmpduSize), std::max (m_voMaxAmpduSize,
m_viMaxAmpduSize))
                                                        + 1.0)
                                                 / std::log (2.0))
                                                - 13.0),
                                        0.0);
     double maxAmpduLengthExponent2 = std::max (std::ceil ((std::log (std::max (std::max
(m_dvoMaxAmpduSize, m_voMaxAmpduSize), std::max (m_kviMaxAmpduSize,
m_dviMaxAmpduSize))
                                                        + 1.0)
                                                 / std::log (2.0))
                                                - 13.0),
                                        0.0);
    double maxAmpduLengthExponent = std::max(maxAmpduLengthExponent1,
maxAmpduLengthExponent2);
```

```
        }
  return capabilities;
}


Ptr<QosTxop>
RegularWifiMac::GetDVOQueue () const
{
  return m_edca.find (AC_DVO)->second;
}


Ptr<QosTxop>
RegularWifiMac::GetKVIQueue () const
{
  return m_edca.find (AC_KVI)->second;
}

Ptr<QosTxop>
RegularWifiMac::GetDVIQueue () const
{
  return m_edca.find (AC_DVI)->second;
}
```

```
  .AddAttribute ("DVO_MaxAmsduSize",
                     UintegerValue (0),
                     MakeUintegerAccessor (&RegularWifiMac::SetDvoMaxAmsduSize),
                     MakeUintegerChecker<uint16_t> (0, 11426))
   .AddAttribute ("VO_MaxAmsduSize",
                UintegerValue (0),
                MakeUintegerAccessor (&RegularWifiMac::SetVoMaxAmsduSize),
                MakeUintegerChecker<uint16_t> (0, 11426))



  .AddAttribute ("KVI_MaxAmsduSize",
                                UintegerValue (0),
                                MakeUintegerAccessor
(&RegularWifiMac::SetKviMaxAmsduSize),
                                MakeUintegerChecker<uint16_t> (0, 11426))
  .AddAttribute ("DVI_MaxAmsduSize",
                                 UintegerValue (0),
                          MakeUintegerAccessor (&RegularWifiMac::SetDviMaxAmsduSize),
                        MakeUintegerChecker<uint16_t> (0, 11426))

 .AddAttribute ("KVI_MaxAmpduSize",
                         UintegerValue (0),
                         MakeUintegerAccessor (&RegularWifiMac::SetKviMaxAmpduSize),
                           MakeUintegerChecker<uint16_t> ())
 .AddAttribute ("DVI_MaxAmpduSize",
               UintegerValue (0),
                   MakeUintegerAccessor (&RegularWifiMac::SetDviMaxAmpduSize),
                  MakeUintegerChecker<uint16_t> ())
```

```
    .AddAttribute ("DVO_BlockAckThreshold",
                    UintegerValue (0),
                    MakeUintegerAccessor (&RegularWifiMac::SetDvoBlockAckThreshold),
                    MakeUintegerChecker<uint8_t> (0, 64))

.AddAttribute ("KVI_BlockAckThreshold",
                UintegerValue (0),
                    MakeUintegerAccessor (&RegularWifiMac::SetKviBlockAckThreshold),
                MakeUintegerChecker<uint8_t> (0, 64))
.AddAttribute ("DVI_BlockAckThreshold",
                                    UintegerValue (0),
                         MakeUintegerAccessor
(&RegularWifiMac::SetDviBlockAckThreshold),
              MakeUintegerChecker<uint8_t> (0, 64))

  .AddAttribute ("DVO_BlockAckInactivityTimeout",
                                   UintegerValue (0),
                                   MakeUintegerAccessor
(&RegularWifiMac::SetDvoBlockAckInactivityTimeout),
           MakeUintegerChecker<uint16_t> ())

   .AddAttribute ("KVI_BlockAckInactivityTimeout",
                       UintegerValue (0),
                       MakeUintegerAccessor
(&RegularWifiMac::SetKviBlockAckInactivityTimeout),
                MakeUintegerChecker<uint16_t> ())

          .AddAttribute ("DVI_BlockAckInactivityTimeout",
            UintegerValue (0),
                                    MakeUintegerAccessor
(&RegularWifiMac::SetDviBlockAckInactivityTimeout),
                              MakeUintegerChecker<uint16_t> ())




   .AddAttribute ("ShortSlotTimeSupported",
                 BooleanValue (true),
                 MakeBooleanAccessor (&RegularWifiMac::SetShortSlotTimeSupported,
                                      &RegularWifiMac::GetShortSlotTimeSupported),
                 MakeBooleanChecker ())

void
RegularWifiMac::ConfigureAggregation (void)
{
  NS_LOG_FUNCTION (this);
  if (GetDVOQueue ()->GetMsduAggregator () != 0)
    {
      GetDVOQueue ()->GetMsduAggregator ()->SetMaxAmsduSize (m_dvoMaxAmsduSize);
    }
  if (GetVOQueue ()->GetMsduAggregator () != 0)
    {
      GetVOQueue ()->GetMsduAggregator ()->SetMaxAmsduSize (m_voMaxAmsduSize);
    }

   if (GetKVIQueue ()->GetMsduAggregator () != 0)
     {
       GetKVIQueue ()->GetMsduAggregator ()->SetMaxAmsduSize (m_kviMaxAmsduSize);
     }
   if (GetDVIQueue ()->GetMsduAggregator () != 0)
     {
       GetDVIQueue ()->GetMsduAggregator ()->SetMaxAmsduSize (m_dviMaxAmsduSize);
     }
  if (GetDVOQueue ()->GetMpduAggregator () != 0)
    {
      GetDVOQueue ()->GetMpduAggregator ()->SetMaxAmpduSize (m_dvoMaxAmpduSize);
    }


    if (GetKVIQueue ()->GetMpduAggregator () != 0)
         {
           GetKVIQueue ()->GetMpduAggregator ()->SetMaxAmpduSize
(m_kviMaxAmpduSize);
         }
    if (GetDVIQueue ()->GetMpduAggregator () != 0)
         {
           GetDVIQueue ()->GetMpduAggregator ()->SetMaxAmpduSize
(m_dviMaxAmpduSize);
         }
         i->second->SetMpduAggregator (mpduAggregator);
       }
    }
  ConfigureAggregation ();
}


}


}
```

Wifi-mac.cc

```
#include "ns3/log.h"
#include "ns3/packet.h"
#include "wifi-mac.h"
#include "txop.h"
#include "ssid.h"

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("WifiMac");

NS_OBJECT_ENSURE_REGISTERED (WifiMac);

  switch (ac)
    {
   case AC_DVO:
      dcf->SetMinCw ((cwmin + 1) / 8 - 1);
      dcf->SetMaxCw ((cwmin + 1) / 4 - 1);
      dcf->SetAifsn (2);
      if (isDsss)
        {
          dcf->SetTxopLimit (MicroSeconds (3264));
        }
      else
        {
```

```
      dcf->SetTxopLimit (MicroSeconds (2080));
    }
  break;



case AC_KVI:
      dcf->SetMinCw ((cwmin + 1) / 4 - 1);
      dcf->SetMaxCw ((cwmin + 1) / 2 - 1);
      dcf->SetAifsn (2);
      if (isDsss)
        {
          dcf->SetTxopLimit (MicroSeconds (6016));
        }
      else
        {
          dcf->SetTxopLimit (MicroSeconds (3264));
        }
      break;

case AC_DVI:
  dcf->SetMinCw ((cwmin + 1) / 4 - 1);
  dcf->SetMaxCw ((cwmin + 1) / 2 - 1);
  dcf->SetAifsn (2);
  if (isDsss)
    {
      dcf->SetTxopLimit (MicroSeconds (6016));
    }
    else
      {
        dcf->SetTxopLimit (MicroSeconds (3264));
      }
    break;




  }
}


}
```

Ap-wifi-mac.cc

```
#include "ns3/log.h"
#include "ns3/packet.h"
#include "ns3/simulator.h"
#include "ns3/pointer.h"
#include "ns3/string.h"
#include "ns3/random-variable-stream.h"
#include "ap-wifi-mac.h"
#include "mac-low.h"
#include "mac-tx-middle.h"
#include "mac-rx-middle.h"
#include "mgt-headers.h"
#include "msdu-aggregator.h"
#include "amsdu-subframe-header.h"
```

```
#include "wifi-phy.h"

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("ApWifiMac");

NS_OBJECT_ENSURE_REGISTERED (ApWifiMac);

    edca = m_edca.find (AC_DVI)->second;
    txopLimit = edca->GetTxopLimit ();
    edcaParameters.SetDviAci (3);
    edcaParameters.SetDviCWmin (edca->GetMinCw ());
    edcaParameters.SetDviCWmax (edca->GetMaxCw ());
    edcaParameters.SetDviAifsn (edca->GetAifsn ());
   edcaParameters.SetDviTxopLimit (static_cast<uint16_t> (txopLimit.GetMicroSeconds () /
32));

    edca = m_edca.find (AC_KVI)->second;
        txopLimit = edca->GetTxopLimit ();
        edcaParameters.SetKviAci (3);
        edcaParameters.SetKviCWmin (edca->GetMinCw ());
        edcaParameters.SetKviCWmax (edca->GetMaxCw ());
        edcaParameters.SetKviAifsn (edca->GetAifsn ());
        edcaParameters.SetKviTxopLimit
(static_cast<uint16_t> (txopLimit.GetMicroSeconds () / 32));



    edca = m_edca.find (AC_VO)->second;
```

```
      txopLimit = edca->GetTxopLimit ();
      edcaParameters.SetVoAci (6);
      edcaParameters.SetVoCWmin (edca->GetMinCw ());
      edcaParameters.SetVoCWmax (edca->GetMaxCw ());
      edcaParameters.SetVoAifsn (edca->GetAifsn ());
      edcaParameters.SetVoTxopLimit (static_cast<uint16_t> (txopLimit.GetMicroSeconds () /
32));

      edca = m_edca.find (AC_DVO)->second;
      txopLimit = edca->GetTxopLimit ();
      edcaParameters.SetDvoAci (7);
      edcaParameters.SetDvoCWmin (edca->GetMinCw ());
      edcaParameters.SetDvoCWmax (edca->GetMaxCw ());
      edcaParameters.SetDvoAifsn (edca->GetAifsn ());
      edcaParameters.SetDvoTxopLimit (static_cast<uint16_t> (txopLimit.GetMicroSeconds () /
32));

    }
  return edcaParameters;
}
```

sta-wifi-mac.cc

```
if (edcaParameters.IsQosSupported ())
        {
          qosSupported = true;
        SetEdcaParameters (AC_BK, edcaParameters.GetBkCWmin (),
edcaParameters.GetBkCWmax (), edcaParameters.GetBkAifsn (), 32 * MicroSeconds
(edcaParameters.GetBkTxopLimit ()));

        SetEdcaParameters (AC_BE, edcaParameters.GetBeCWmin (),
edcaParameters.GetBeCWmax (), edcaParameters.GetBeAifsn (), 32 * MicroSeconds
(edcaParameters.GetBeTxopLimit ()));

        SetEdcaParameters (AC_VI, edcaParameters.GetViCWmin (),
edcaParameters.GetViCWmax (), edcaParameters.GetViAifsn (), 32 * MicroSeconds
(edcaParameters.GetViTxopLimit ()));

        SetEdcaParameters (AC_DVI, edcaParameters.GetDviCWmin (),
edcaParameters.GetDviCWmax (), edcaParameters.GetDviAifsn (), 32 * MicroSeconds
(edcaParameters.GetDviTxopLimit ()));

        SetEdcaParameters (AC_KVI, edcaParameters.GetKviCWmin (),
edcaParameters.GetKviCWmax (), edcaParameters.GetKviAifsn (), 32 * MicroSeconds
(edcaParameters.GetKviTxopLimit ()));

        SetEdcaParameters (AC_VO, edcaParameters.GetVoCWmin (),
edcaParameters.GetVoCWmax (), edcaParameters.GetVoAifsn (), 32 * MicroSeconds
(edcaParameters.GetVoTxopLimit ()));

        SetEdcaParameters (AC_DVO, edcaParameters.GetDvoCWmin (),
edcaParameters.GetDvoCWmax (), edcaParameters.GetDvoAifsn (), 32 * MicroSeconds
(edcaParameters.GetDvoTxopLimit ()));


    }


if (edcaParameters.IsQosSupported ())

    {

        qosSupported = true;
```

```
            SetEdcaParameters (AC_BK, edcaParameters.GetBkCWmin (),
edcaParameters.GetBkCWmax (), edcaParameters.GetBkAifsn (), 32 * MicroSeconds
(edcaParameters.GetBkTxopLimit ()));

      SetEdcaParameters (AC_BE, edcaParameters.GetBeCWmin (),
edcaParameters.GetBeCWmax (), edcaParameters.GetBeAifsn (), 32 * MicroSeconds
(edcaParameters.GetBeTxopLimit ()));

      SetEdcaParameters (AC_VI, edcaParameters.GetViCWmin (),
edcaParameters.GetViCWmax (), edcaParameters.GetViAifsn (), 32 * MicroSeconds
(edcaParameters.GetViTxopLimit ()));

      SetEdcaParameters (AC_DVI, edcaParameters.GetDviCWmin (),
edcaParameters.GetDviCWmax (), edcaParameters.GetDviAifsn (), 32 * MicroSeconds
(edcaParameters.GetDviTxopLimit ()));

      SetEdcaParameters (AC_KVI, edcaParameters.GetKviCWmin (),
edcaParameters.GetKviCWmax (), edcaParameters.GetKviAifsn (), 32 * MicroSeconds
(edcaParameters.GetKviTxopLimit ()));

      SetEdcaParameters (AC_VO, edcaParameters.GetVoCWmin (),
edcaParameters.GetVoCWmax (), edcaParameters.GetVoAifsn (), 32 * MicroSeconds
(edcaParameters.GetVoTxopLimit ()));

      SetEdcaParameters (AC_DVO, edcaParameters.GetDvoCWmin (),
edcaParameters.GetDvoCWmax (), edcaParameters.GetDvoAifsn (), 32 * MicroSeconds
(edcaParameters.GetDvoTxopLimit ()));


    }
```

Scenario.cc

```
#include "ns3/command-line.h"
#include "ns3/log.h"
```

```
#include "ns3/yans-wifi-helper.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/wifi-module.h"
#include "ns3/ssid.h"
#include "ns3/string.h"
#include "ns3/pointer.h"
#include "ns3/boolean.h"
#include "ns3/double.h"
#include "ns3/mobility-helper.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/udp-client-server-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/wifi-net-device.h"
#include "ns3/qos-txop.h"
#include "ns3/uinteger.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/wifi-mac.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"


using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("he-wifi-network");
uint32_t payloadSize = 3000;

int main (int argc, char *argv[])
{
```

```
          double simulationTime = 20; //seconds
         uint32_t VoSta = 1;
          uint32_t ViSta = 1;
          uint32_t BeSta = 1;
          uint32_t BkSta = 1;

          CommandLine cmd;
          cmd.Parse (argc, argv);

    NodeContainer VoStation;
    VoStation.Create (VoSta);
          NodeContainer ViStation;
          ViStation.Create (ViSta);
          NodeContainer BeStation;
          BeStation.Create (BeSta);
          NodeContainer BkStation;
          BkStation.Create (BkSta);
          NodeContainer Ap;
          Ap.Create (1);
    //Yet Another Network Simulator = yans
          YansWifiPhyHelper PhyHelper = YansWifiPhyHelper::Default ();
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
          PhyHelper.SetChannel (channel.Create ());
          PhyHelper.Set ("GuardInterval", TimeValue(NanoSeconds (1600
          WifiHelper Wifi;
          Wifi.SetStandard (WIFI_PHY_STANDARD_80211ax_5GHZ);
```

```
   Wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode", StringValue ("HeMcs11"),                                    "ControlMode",
StringValue ("HeMcs0"));
    Ssid ssid = Ssid ("AP-80211AX");



    WifiMacHelper MacVoSta;
    MacVoSta.SetType ("ns3::StaWifiMac",
    "HeSupported", BooleanValue (true),
    "QosSupported", BooleanValue (true),
    "Ssid", SsidValue (ssid));

    WifiMacHelper MacViSta;
    MacViSta.SetType ("ns3::StaWifiMac",
    "HeSupported", BooleanValue (true),
    "QosSupported", BooleanValue (true),
    "Ssid", SsidValue (ssid));


     WifiMacHelper MacBeSta;
     MacBeSta.SetType ("ns3::StaWifiMac",
     "HeSupported", BooleanValue (true),
     "QosSupported", BooleanValue (true),
     "Ssid", SsidValue (ssid));

     WifiMacHelper MacBkSta;
     MacBkSta.SetType ("ns3::StaWifiMac",
    "HeSupported", BooleanValue (true),
    "QosSupported", BooleanValue (true),
    "Ssid", SsidValue (ssid));



    WifiMacHelper MacAp;
    MacAp.SetType ("ns3::ApWifiMac", //Handle association, dis-association and
authentication, of STAs within an infrastructure BSS.
    "QosSupported", BooleanValue (true),
            "BeaconGeneration", BooleanValue (true),
            "BeaconInterval", TimeValue (Seconds (2.5)),
    "Ssid", SsidValue (ssid));



    NetDeviceContainer VoNetDev;
    VoNetDev = Wifi.Install (PhyHelper, MacVoSta, VoStation);
    NetDeviceContainer ViNetDev;
    ViNetDev = Wifi.Install (PhyHelper, MacViSta, ViStation);
    NetDeviceContainer BeNetDev;
    BeNetDev = Wifi.Install (PhyHelper, MacBeSta, BeStation);
    NetDeviceContainer BkNetDev;
    BkNetDev = Wifi.Install (PhyHelper, MacBkSta, BkStation);



   NetDeviceContainer ApNetDev;
   ApNetDev = Wifi.Install (PhyHelper, MacAp, Ap);
```

```
 MobilityHelper StaMobility;

StaMobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                   "MinX", DoubleValue (0.0),
                                   "MinY", DoubleValue (0.0),
                                   "DeltaX", DoubleValue (2.0),
                                   "DeltaY", DoubleValue (2.0),
                                   "GridWidth", UintegerValue (4),
                                   "LayoutType", StringValue ("RowFirst"));





 StaMobility.SetMobilityModel ("ns3::RandomDirection2dMobilityModel",
                              "Bounds", RectangleValue (Rectangle (0, 100, 0, 100)),
                              "Speed", StringValue ("ns3::ConstantRandomVariable[Constant=2]"),
                              "Pause", StringValue ("ns3::ConstantRandomVariable[Constant=2.0]"));
 StaMobility.Install (VoStation);
 StaMobility.Install (ViStation);
 StaMobility.Install (BeStation);
 StaMobility.Install (BkStation);

 MobilityHelper ApMobility;
 Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>();
 positionAlloc->Add (Vector ( 3 , 3 , 2));
 ApMobility.SetPositionAllocator (positionAlloc);
ApMobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
ApMobility.Install (Ap);


  InternetStackHelper stack;
  stack.Install (Ap);
   stack.Install (VoStation);

  stack.Install (ViStation);
  stack.Install (BeStation);
  stack.Install (BkStation);


  Ipv4AddressHelper address;
  address.SetBase ("192.168.1.0", "255.255.255.0");


  Ipv4InterfaceContainer ApInt;
  ApInt = address.Assign (ApNetDev);

  Ipv4InterfaceContainer VoStaInt;
  VoStaInt = address.Assign (VoNetDev);
  Ipv4InterfaceContainer ViStaInt;
  ViStaInt = address.Assign (ViNetDev);
  Ipv4InterfaceContainer BeStaInt;
  BeStaInt = address.Assign (BeNetDev);
  Ipv4InterfaceContainer BkStaInt;
  BkStaInt = address.Assign (BkNetDev);
  Ipv4GlobalRoutingHelper::PopulateRoutingTables (); //Helper class that adds
ns3::Ipv4GlobalRouting objects.


  // Setting applications
    //AC_BE (112-0x70), AC_BK (40-0x28), AC_VI (184-0xB8), AC_VO (192-0xC0)


  uint16_t port = 9;
  UdpServerHelper server (port);
  ApplicationContainer serverApp = server.Install (Ap); // Configured in AP
  serverApp.Start (Seconds (0.0));
  serverApp.Stop (Seconds (simulationTime + 1));


   UdpClientHelper client (ApInt.GetAddress (0), port);
   client.SetAttribute ("MaxPackets", UintegerValue (10000));
   client.SetAttribute ("Interval", TimeValue (Time ("0.0002")));
   client.SetAttribute ("PacketSize", UintegerValue (payloadSize));
  ApplicationContainer clientAppsA;
  clientAppsA.Add (client.Install (VdStation));
  InetSocketAddress destA1 (VdStaInt.GetAddress (0), port);
  destA1.SetTos (0xe0); //AC_DVO


  ApplicationContainer clientAppsB;
  clientAppsB.Add (client.Install (VoStation));
  InetSocketAddress destB1 (VoStaInt.GetAddress (0), port);
  destB1.SetTos (0xc0); //AC_VO



   ApplicationContainer clientAppsC;
```

```
clientAppsC.Add (client.Install (VkStation));
InetSocketAddress destC1 (VkStaInt.GetAddress (0), port);
destC1.SetTos (0xa0); //AC_VI

ApplicationContainer clientAppsD;
clientAppsD.Add (client.Install (VcStation));
InetSocketAddress destD1 (VcStaInt.GetAddress (0), port);
destD1.SetTos (0x80); //AC_VI



ApplicationContainer clientAppsE;
clientAppsE.Add (client.Install (ViStation));
InetSocketAddress destE1 (ViStaInt.GetAddress (0), port);
destE1.SetTos (0x40); //AC_VI

ApplicationContainer clientAppsF;
clientAppsF.Add (client.Install (BeStation));
InetSocketAddress destF1 (BeStaInt.GetAddress (0), port);
destF1.SetTos (0x02); //AC_BE

ApplicationContainer clientAppsG;
clientAppsG.Add (client.Install (BkStation.Get(0)));
InetSocketAddress destG1 (BkStaInt.GetAddress (0), port);
destG1.SetTos (0x20); //AC_BK  ----

ApplicationContainer clientAppsH;
clientAppsH.Add (client.Install (BkStation.Get(1)));
InetSocketAddress destH1 (BkStaInt.GetAddress (1), port);
```

```
    destH1.SetTos (0x20); //AC_BK  ----

    clientAppsA.Start (Seconds (1.0));
     clientAppsB.Start (Seconds (1.0));
      clientAppsC.Start (Seconds (1.1));
      clientAppsD.Start (Seconds (1.5));
      clientAppsE.Start (Seconds (1.0));
      clientAppsF.Start (Seconds (1.0));
      clientAppsG.Start (Seconds (9.0));
      clientAppsH.Start (Seconds (10.0));



      clientAppsA.Stop (Seconds (4.0));
      clientAppsB.Stop (Seconds (10.0));
      clientAppsC.Stop (Seconds (12.0));
      clientAppsD.Stop (Seconds (13.5));
      clientAppsE.Stop (Seconds (16.0));
      clientAppsF.Stop (Seconds (11));
      clientAppsG.Stop (Seconds (19.0));
      clientAppsH.Stop (Seconds (20.0));

      Simulator::Stop (Seconds (simulationTime + 1));

      AnimationInterface anim ("my-animation.xml");.

      anim.SetMaxPktsPerTraceFile(10000);


       Ptr<FlowMonitor> monitor = flowmon.InstallAll (); //An object that monitors and reports
back packet flows observed during a simulation.

       PhyHelper.EnablePcapAll ("my");

       Simulator::Run ();

       monitor->CheckForLostPackets ();

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmon.GetClassifier ()); //Provides a method to translate raw packet data into abstract flow
identifier and packet identifier parameters.

       FlowMonitor::FlowStatsContainer stats = monitor-> GetFlowStats (); //Container:
FlowId, FlowStats.


       uint32_t txPacketsum = 0;

       uint32_t rxPacketsum = 0;
```

```
        uint32_t DropPacketsum = 0;

        uint32_t LostPacketsum = 0;

        double Delaysum = 0;


        for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i !=
stats.end (); ++i)
         {
           txPacketsum += i->second.txPackets;

           rxPacketsum += i->second.rxPackets;

           LostPacketsum += i->second.lostPackets;

           DropPacketsum += i->second.packetsDropped.size();

           Delaysum += i->second.delaySum.GetSeconds();

         }


        std::cout << "  All Tx Packets: " << txPacketsum << "\n";

        std::cout << "  All Rx Packets: " << rxPacketsum << "\n";

        std::cout << "  All Delay: " << Delaysum / txPacketsum <<"\n";

        std::cout << "  All Lost Packets: " << LostPacketsum << "\n";

        std::cout << "  All Drop Packets: " << DropPacketsum << "\n";

        std::cout << "  Packets Delivery Ratio: " << ((rxPacketsum *100) / txPacketsum) << "%"
<< "\n";

        std::cout << "  Packets Lost Ratio: " << ((LostPacketsum *100) / txPacketsum) << "%"
<< "\n";

        std::cout <<"Throughput: " << rxPacketsum * payloadSize * 8 / (10 * 1000000.0)<<
"mbps" << "\n";




        monitor->SerializeToXmlFile("myflow.xml", true, true);
```

```
    Simulator::Destroy ();

  return 0;
}
```