Implementation and Performance Analysis of rate adaptation algorithms for WLANs

Dinesh Kumar Maurya (B14BS005)
Vijay Kumar Paliwal (B14CS040)

I. Abstract

IEEE 802.11 WLAN is the most used wireless system in practical applications. The wireless medium suffers through many disturbances and hence the wireless transmission is affected. Our aim is to maximize the throughput of the system considering all channel conditions. The physical layer in WLANs provide multiple link rates for transmission. There might be issues with both high and low rates, so the rate of transmission is chosen so as to maximize the throughput. There has been a lot of work going on developing several algorithms for determination of rates on MAC layer, but there is little literature available providing complete comparison based on performance analysis of these algorithms. In this project we present the implementation and performance comparison of two such rate adaptation algorithms. We have implemented the algorithms in NS-2.35 (a network simulator). The comparison of performance is done based on some wireless scenarios generated in the simulator. The major parameter indicating the performance of the algorithm is total throughput of the system. We have considered two major kind of scenarios. One was with three nodes and one access point and another was having 8 nodes and one access point. First scenario had less collision probability so the performance of both of the algorithms was almost same. But in the second scenario, which is more close to practical scenarios, the throughput of AARF algorithm was more than 40 percent better than the ARF algorithm.

II. Problem Statement

We have implemented two rate adaptation algorithms in MAC layer for IEEE 802.11 networks, namely : 1) Auto Rate Fallback (ARF) Algorithm 2) Adaptive Auto Rate Fallback (AARF) Algorithm After implementing the two algorithms wireless scenarios were generated for simulation of practical network conditions and testing and calculation of throughput of the two algorithms. The network simulator NS-2.35 is used for implementation and simulation of the above mentioned algorithms.

III. Introduction

IEEE 802.11 networks are wireless local area networks. The transmission of data in wireless network depends on several factors. Some of them are the channel conditions, number of users/nodes, collisions of packets. These all factors influence the transmission of data and hence the total throughput of the system. The physical layer provides multiple rates for transmission of data. Consider an example, a machine is transmitting data at rate r0. Now if channel conditions get worse or better, the throughput can be increased by changing the data transmission rate. For example if the rate is kept high even in bad channel conditions, then all transmissions may fail and hence the efficiency will be low. On the other hand if the rate is not increased in case of better channel conditions, the transmission might not be as efficient as much it could have been at higher rate. The rate for transmission of data can be controlled based on several factors to maximize the throughput of the system. So there is a requirement of smart algorithms which can control the link rate. The determination is done dynamically on MAC layer, so as to respond for a live system. Therefore a rate adaptation algorithm is a model which takes several parameters as signal to noise ratio, number of successfully transmitted packets, number of failed packets etc. as input and determine the rate of link as its output. There has been a lot of work in the field of the rate adaptation algorithms. Each algorithms focusses on an aspect of issues related to wireless transmission, but as mentioned earlier, there has not been any complete study on performance comparison of all these algorithms. This kind of analysis is required so as to get direction for further work in the field of developing new rate adaptation algorithms. In this project, we have implemented two basic rate adaptation algorithms ARF and AARF. These two algorithms are basic and differ in the way how they tackle the failure in transmission of packets and determination of the rate. So considering practical cases, where collisions is a major issue, we decided to implement these two algorithms and to compare their performance. We assume that output of this project will be guide the researchers in finding solutions for issues related to existing algorithms. Also this kind of project will further become a motivation for further work in the same direction.

IV. Literature survey

Rate adaptation in IEEE 802.11 networks is a well known and highly studied topic. There have been plenty of algorithms proposed in literature, although some of them can not be implemented in real scenarios. In this section, we describe the most known algorithms ARF and AARF which we have implemented. ARF[1] is a widely adopted and well known algorithm. This

algorithm is acknowledgement based simple model. The decision of increasing or decreasing the rate is based on number of consecutively successfully transmitted packets or number of failed packets. If 10 consecutive packets are successfully transmitted (or 10 consecutively ack packets are received) then it attempts to increase the rate to next level. If the packet fails to be transmitted successfully, immediately after increasing the rate, it decreases the rate back to previous level. This algorithm is widely adopted because of being simple. The drawback of this algorithms is that is fails to identify difference between the reasons of collisions. Another drawback is that it attempts to increase the rate after every ten successful transmission, even if current rate is the most convenient. AARF[1] is a modified version of ARF algorithm. It mainly deals with the second issue related to the ARF approach. The decision of increasing or decreasing the rate is again based on number of consecutively successfully transmitted packets, but this threshold number is not constant (as it was 10 in ARF). Indeed, if the first packet after increasing rate fails to transmit, then we double the threshold number, i.e. the next attempt to increase the rate will happen after 20 successful transmissions. Again if the first packet fails after attempt to increase the rate, if again increases the threshold. There is a maximum limit on this threshold (160 in our implementation). If the attempt to increase the rate succeeds, the threshold is reset to basic value (10 in this case). This algorithm gives better throughput in multi-user environment, because it tries to stabilize at most suitable rate.Also this paper describes that there has been no analysis for comparison of different rate adaptation algorithms. Also there is a need of an algorithm which can quickly respond to transient channel conditions as well as collisions. NS2[2] is a discrete event simulator for research in networking. It provides support for simulation of routing, TCP and various multicast protocols for wired and wireless networks. NS2 is written in C++ and TCL as scripting language. It also supports graphical representation of network topologies and transmission of data. We have used NS2.35 for our project. The implementation of rate adaptation algorithms was done in C++ in MAC layer. The generation of wireless scenario for testing of algorithms and evaluation of throughput of the algorithms was done in TCL script. We have also studied a paper about fingerprinting of rate adaptation algorithms, which has helped in determining some of key factors covering the performance analysis of rate adaptation algorithms[3].

V. Methodology

There were mainly three stages in this project. First stage includes understanding the concept and working on wireless network and understanding of rate adaptation algorithms. Second stage includes understanding the simulator and implementation of the algorithms and the third stage was simulation of test scenario with TCL scripts and performance evaluation of the implemented algorithms.

A. Understanding concepts of wireless networks and rate adaptation algorithms

The IEEE 802.11 networks are wireless local area networks. The medium of transmission is wireless, hence it is shared among all nodes who are competing accessing the resource/medium for transmitting, so Medium Access Control protocol is used for accessing the medium. IEEE 802.11 includes a frame exchange protocol at MAC layer for confirmation of successful transmission of data. This mechanism includes the sending and receiving of RTS (request to send), CTS (clear to send) and ACK (acknowledgement). The data transfer mechanism in IEEE 802.11 includes exchange of four frames. First the source sends the RTS, then the destination sends CTS. After receiving CTS, the source sends data, then destination sends ACK, which confirms successful transmission of data. In case the ACK is not received by the source a request to retransmit data is generated. The timing for sending or waiting for access to medium is determined by DCF (Distributed Coordination Function). Further the physical layer in most of IEEE 802.11 networks supports multiple rates for transmission of data. So we can use this facility to increase throughput of the system. Therefore various algorithms have been proposed for controlling the rate based on several factors. Two of these algorithms are ARF and AARF, which we have implemented in this project. The flow chart for ARF and AARF are as given in figure 1 and 2.

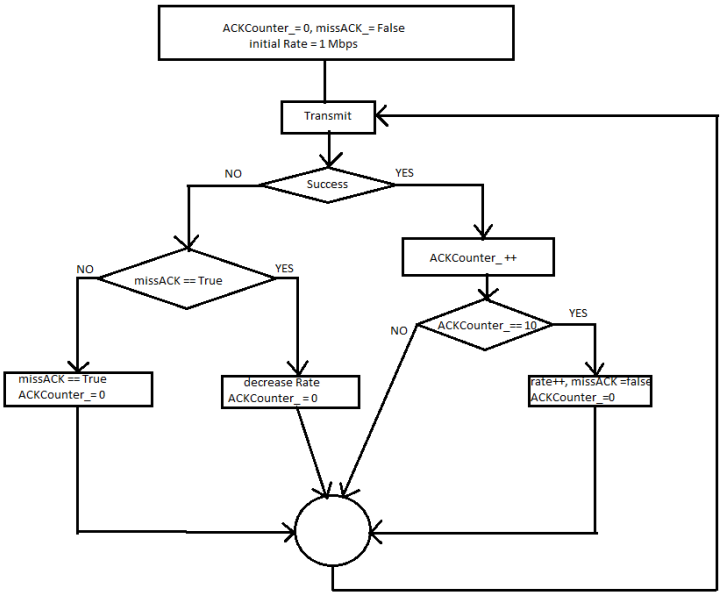

Fig1: ARF Algorithm

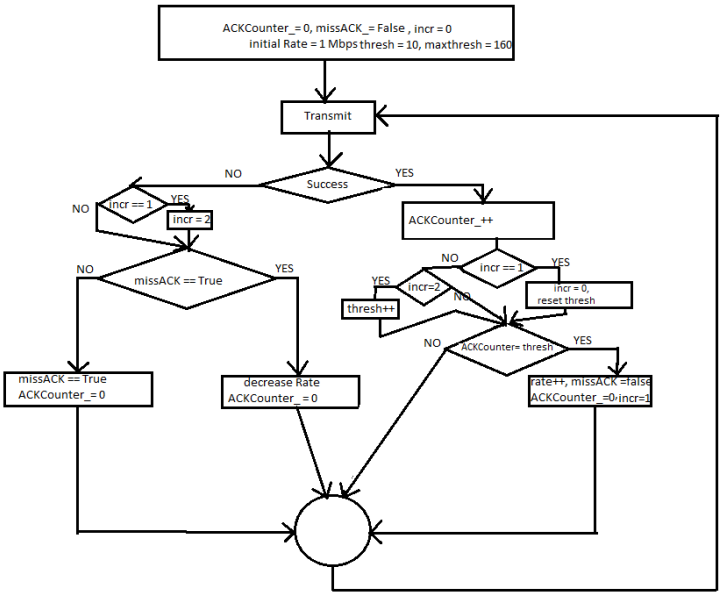

Fig2: AARF Algorithm

B. Implementing the algorithms

The algorithms have been implemented in NS-2.35. We have Mac802.11 class in mac layer in ns2. Mainly we have changed the two functions in Mac802.11 class which is inherited from Mac class. As mentioned earlier, the successful transmission of data is detected when ACK is received and failure is detected when request is generated to retransmit data. So we have changed the recvACK (Packet *) and RetransmitDATA ( ) function in Mac802.11 class, which is implemented in ns-2.35/mac/mac-802.11.cc file.

The changed code proportion where the algorithm is implemented is as follows:

Implemented ARF:

In recvACK (Packet *) function:

missACK$_=false$;

ACKcounter$_=ACKcounter_+1$;

if (ACKcounter$_>10$)

//upgrade datarate

int m;

for (m=0;m¡4;m++)

if (optRate$_{[}m] == dataRate_{m!=3)}$
dataRate$_{=}optRate_{[}m + 1]$;
break;
ACKcounter$_{=}0$;
In RetransmitDATA ( ) function:
ACKcounter$_{=}0; //nosuccessiveACKs$
if (missACK$_{=} = false)$
missACK$_{=}true; //firstmissedACKdoesnottriggerfallback$
else if (dataRate$_{=} = basicRate_{)}$
//already fallback to lowest
else
//previous rate is datarate$_{a}ssameasgetTxRate$
ch-¿size() -= phymib$_{.}getHdrLen11(pktTx_{-} > txinfo_{.}getTxRate())$;
// assert pktTx and change size
int m;
for (m=0;m¡4;m++)
if (optRate$_{[}m] == dataRate_{m!=0)}$
dataRate$_{=}optRate_{[}m - 1]$;
//calculate new size
ch-¿size() += phymib$_{.}getHdrLen11(dataRate_{)}$;
ch-¿txtime() = txtime(ch-¿size(), dataRate$_{)}$;
pktTx$_{-} > txinfo_{.}setTxRate(dataRate_{)}$;
2. Implemented AARF:
In recvACK (Packet *) function:
missACK$_{=}false$;
int thresh[5];
thresh[0] = 10;
thresh[1] = 20;
thresh[2] = 40;
thresh[3] = 80;
thresh[4] = 160;
if(incr==2)
incr = 0;
if(arr$_{i}ndex < 4)$
arr$_{i}ndex + +$;
if(incr==1)
arr$_{i}ndex = 0$;
ACKcounter$_{=}ACKcounter_{+}1$;
if (ACKcounter$_{>}thresh[arr_{i}ndex])$
//upgrade datarate
int m;
for (m=0;m¡4;m++)
if (optRate$_{[}m] == dataRate_{m!=3)}$
dataRate$_{=}optRate_{[}m + 1]$;
incr = 1;
break;
ACKcounter$_{=}0$;
In RetransmitDATA ( ) function:
if(incr==1)
failed = 1;
incr = 2;
ACKcounter$_{=}0; //nosuccessiveACKs$
if (missACK$_{=} = false)$
missACK$_{=}true; //firstmissedACKdoesnottriggerfallback$
else if (dataRate$_{=} = basicRate_{)}$
//already fallback to lowest
else
//previous rate is datarate$_{a}ssameasgetTxRate$

ch-¿size() -= phymib$_{.}getHdrLen11(pktTx_{-} > txinfo_{.}getTxRate());$
// assert pktTx and change size
int m;
for (m=0;m¡4;m++)
if (optRate$_{[}m] == dataRate_{m!=0)}$
dataRate$_{=}optRate_{[}m-1];$
//calculate new size
ch-¿size() += phymib$_{.}getHdrLen11(dataRate_{)};$
ch-¿txtime() = txtime(ch-¿size(), dataRate$_{)};$
pktTx$_{-} > txinfo_{.}setTxRate(dataRate_{)};$

C. *Simulation and testing*

Simulations of the algorithm were conducted in NS-2.35 using TCL script.

VI. Experiments and Results:

The topology used for the performance measurement of the algorithms was consisting of one AP and eight nodes. On same script both the algorithms were run and the throughput was measured. The configuration of nodes was as shown in figure 3 and figure 4.

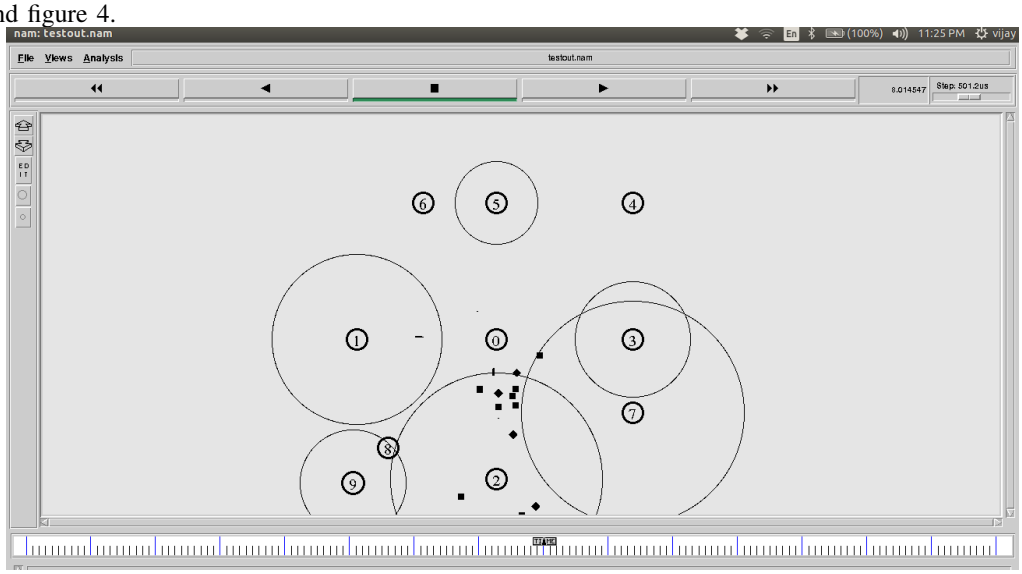

Fig3: Test Scenario

The throughput measured from previous test is as given in figure 4 and figure 5.

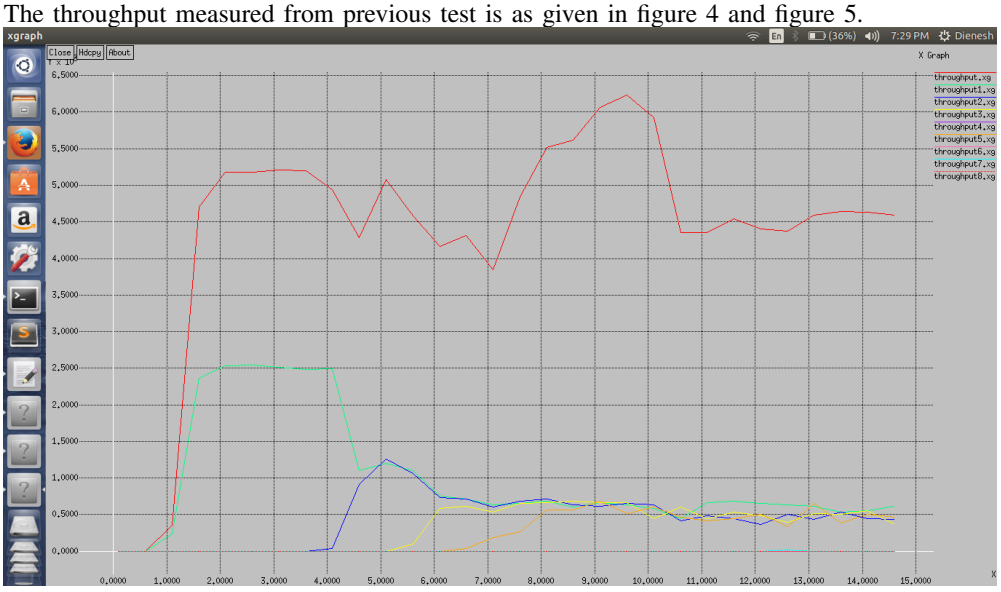

Fig4: ARF Throughput

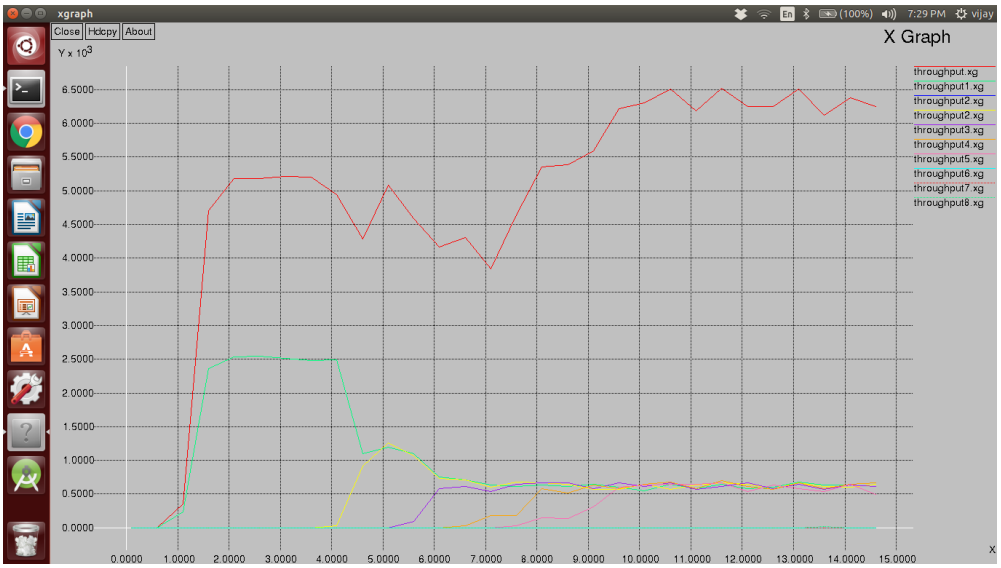

Fig5: AARF Throughput

VII. Conclusions:

We see that after 10 seconds in the simulation, when the throughput gets stabilized, the AARF gives around 6.25 MBps as average throughput, while the ARF gives 4.4MBps as average throughput. So we see that AARF has increased throughput up to more than 42 percent better performance in higher congested network conditions. Although in scenarios where less number of users are present, both have almost same throughput, but first case is more close to practical cases. Therefore, we conclude that the algorithm must be robust enough to stand at an intermediate which is much convenient. Also run time learning of environment and reacting helps us respond to network conditions in a better way and hence increases the ability to maximize throughput. We hope that taking these facts in consideration, further researchers will be able to develop new algorithms in better way.

VIII. References

1. Saiid Biaz and Shaoen Wu, Rate Adaptation Algorithms for IEEE 802.11 Networks: A Survey and Comparison, in Proc. IEEE Symposium on Computers and Communication, pp. 130-136, 2008

2. NS2, http://www.isi.edu/nsnam/ns 2006. [Online]. Available: ”http://www.isi.edu/nsnam/ns/”

3. M.Mirza, P Barford, X Zhu and S Banerjee, Fingerprinting 802.11 Rate Adaptation Algorithms, in Proc. INFOCOM, 2011, pp. 1-9, 2011