Int. J. Communications, Network and System Sciences, 2009, 4, 249-324
doi:10.4236/ijcns.2009.24033 Published Online July 2009 (http://www.SciRP.org/journal/ijcns/).

Adaptive Backoff Algorithm for IEEE 802.11 MAC Protocol

Maali ALBALT[1], Qassim NASIR[2]
Department of Electrical and Computer Engineering, College of Engineering, University of Sharjah, Sharjah, UAE
Email: {[1]maali, [2]nasir}@sharjah.ac.ae
Received October 4, 2008; *revised December* 28, 2008*; accepted March* 3, 2009

ABSTRACT

A Mobile Ad Hoc Network (MANET) is a collection of mobile nodes that can communicate directly over wireless media, without the need for a pre-configured infrastructure. Several approaches have been suggested to improve Quality of Service (QoS) in IEEE 802.11-based MANETs through modifying some of the IEEE 802.11 Medium Access Control (MAC) algorithms, such as the backoff algorithm that is used to control the packets collision aftermath. In this work, an adaptive IEEE 802.11 backoff algorithm to improve QoS is developed and tested in simulations as well as in testbed implementation. While the Binary Exponential Backoff (BEB) algorithm deployed by IEEE 802.11 reacts based on individual packet transmit trials, the new algorithm takes the history of successive packet transmit trials into account to provide a better QoS performance.

The new algorithm has been tested against the legacy IEEE 802.11 through simulations using QualNet and a Linux-based testbed comprising a number of stations. The performed tests have shown significant improvements in performance, with up to 33.51% improvement in delay and 7.36% improvement in packet delivery fraction compared to the original IEEE 802.11.

Keywords: MANETs, Ad-Hoc Networks, Quality of Service, Backoff Algorithm, IEEE 802.11

1. Introduction

A MANET is a collection of mobile nodes that are connected without any infrastructure or base station [1]. In such networks, nodes are free to enter, leave the network, move and organize themselves; thus, the topology of the network can change unpredictably. Since the wireless medium is shared by all transmitting nodes in range, there should be a mechanism to control medium access among contending stations so as to minimize the effect of collisions on the performance of the network. The famous IEEE 802.11, for instance, adopts a binary exponential backoff (BEB) algorithm that exponentially increases a station's waiting time if the medium is found busy, and resets to a minimum value right after a successful transmission. The BEB algorithm is considered "memory-less" since it resets the Contention Window (CW) value to the minimum right after a successful transmission without taking into consideration the network conditions.

Many researchers were motivated to enhance the performance of the IEEE 802.11 through modifying the BEB algorithm [1–12]. Most of the prior work in this area have changed or modified the BEB algorithm such that it provides relative priority among two or more traffic classes [1–5]. This solves the *intra-class* contention problem since the class with the least specified CW value would access the channel first. However, it does not solve the *inter-class* contention problem since a number of stations wishing to send packets of the same priority class may still contend and collide (in case their backoff timers expire simultaneously). The latter problem is solved by determining what to do until a successful transmission takes place, or simply how to increment the value of the contention window in the case of a busy channel. The suggested backoff algorithms were mostly slight variations or scales of the BEB algorithm for each traffic class. However, the way how to decrement the CW was unaddressed explicitly and hence assumed to remain the same as the original BEB algorithm (the CW would reset to CWmin upon a successful transmission).

It can be said that the above works solved the contention problem from a priority point of view through determining which class should access the medium first.

But that is not enough since the sudden CW reset to CWmin may cause several collisions, which requires addressing the contention problem from a congestion point of view. More specifically, when a station succeeds in transmitting a packet at a given CW, that doesn't mean a decrease in congestion, but it means arriving at a convenient CW value [6].

This finding inspired some researchers to adopt a different approach of looking at the backoff algorithm, which is what to do after a successful transmission takes place, or simply how to decrement the CW. Several works on slowly decreasing the CW value were proposed [6–12]. [7] and [8] suggested an exponential decrease in the CW value upon a successful transmission instead of resetting to CWmin, but they assumed a fixed scale of decrease without taking the network conditions into account. That might result in underutilizing the channel if it was idle or returning to a congestion state if the network had not yet relieved from a previous congestion.

While [6] and [9] suggested slow decrease backoff algorithms to adapt to the network load, there was another proposal to assume a p-persistent MAC protocol in which the station would transmit with a probability p and refrain form transmitting with a probability 1-p [10,11]. That p-value was calculated in runtime and updated after each transmission to reflect the current number of active stations in [10] or the average time the channel is idle or busy in [11] among other conditions that affect the network load. In both the slow decrease and p-persistent cases, complex computations were needed to update the p-value and to estimate the network load, respectively. Complex computations also mean high power consumption, which is in many cases considered unaffordable in the wireless ad hoc networks context.

This work has modified the IEEE 802.11's main mechanism for managing access to the shared wireless medium, which is the Binary Exponential Backoff (BEB) algorithm; it is replaced with a History-Based Adaptive Backoff (HBAB) algorithm that updates the value of the BEB's Contention Window (CW) according to the network conditions with a suitable prediction. The algorithm proposes a novel approach to slowly increase and decrease the CW value based on the busyness of the channel, i.e. MAC layer transmission retrials. In a previous work by the same authors [13] a similar backoff algorithm was introduced under the same name, but it used a different technique to update CW. Besides developing HBAB in theory and testing it in simulation, this work has built a Linux-based MANET to act as a testbed for implementing HBAB. The constructed testbed, which integrates a number of ready-made and customized hardware and software components, implements HBAB as well as the original IEEE 802.11 protocols in real-time. Both simulation and testbed results show significant improvements in QoS related performance measures, especially in delay, when using HBAB over the original IEEE 802.11.

The paper is organized as follows. Section 2 reviews the IEEE 802.11 MAC protocol backoff algorithm and QoS in MANETs. Section 3 presents the HBAB algorithm. Sections 4 and 5 discuss performance evaluation of the proposed HBAB against the standard BEB IEEE 802.11 in simulation and Linux testbed, respectively. Section 6 concludes the paper and provides directions for future work.

2. IEEE 802.11 and Binary Exponential Backoff

The IEEE 802.11 family of standards defines the specifications of both the physical (PHY) and medium access control (MAC) layers to construct a WLAN [14]. While the 802.11 PHY layer defines the signaling and modulation properties of the protocol, the 802.11 MAC layer controls access to the shared wireless medium. In order to accomplish that, the 802.11 MAC defines two medium access functions: a mandatory distributed coordination function (DCF) and an optional point coordination function (PCF) [3]. The DSF function uses BEB to manage access to the medium in the case of packet collisions.

The DCF function uses a carrier sense multiple access with collision avoidance (CSMA/CA) mechanism to control access to the shared wireless medium. This mechanism features the exchange of control packets (Request-To-Send (RTS) and Clear-To-Send (CTS)) before data transmission to minimize the chances of collisions. Furthermore, before initiating that type of RTS/CTS exchange, each STA is required to sense the medium for a time interval consisting of the DCF Interframe Space (DIFS) and the current value of the backoff timer [3]. The value of the backoff timer is randomly picked in the range of the current Contention Window (CW) of the station. CW value is updated by a Binary Exponential Backoff (BEB) algorithm that exponentially increases a station's waiting time if the medium is found busy, and resets to a minimum value right after a successful transmission. A positive Acknowledgment (ACK) is used to notify the sender that the frame has been successfully received. If an ACK is not received within a time period of ACKTimeout, the sender assumes that there is a collision and schedules a retransmission by entering the backoff process again until the maximum retransmission limit is reached [3]. A maximum of 7 retransmissions (4 retransmissions) for short frames (long frames) are allowed before the frame is dropped. The basic access procedure is shown in Figure 1.

The BEB algorithm is used by the IEEE 802.11 to control access to the shared wireless medium among contending stations. This is done through adjusting the contention window size based on the current medium status. When a station has some data to send, it senses the channel to determine whether it is idle. If the medium remains idle for a time interval equal to DIFS, the station is allowed to transmit. If the medium is busy, the transmission is postponed until the ongoing transmission concludes. Meanwhile, a slotted binary exponential backoff procedure takes place: each slot is equal to DIFS, and the number of such slots is determined by a random value uniformly chosen in [0, CW -1], where CW is the current contention window size.

That random value is used to initialize the backoff timer, which keeps running as long as the channel is sensed idle, paused when data transmission (initiated by other stations) is in progress, and resumed when the channel is sensed idle again for more than DIFS. The time immediately following an idle DIFS is slotted, with each slot equal to the time needed for any station to detect the transmission of a frame (in the IEEE 802.11 term, MAC Service Data Unit (MSDU)) from any other station. When the backoff timer expires, the station attempts to transmit a data frame at the beginning of next slot.

Finally, if the data frame is successfully received, the receiver transmits an acknowledgment frame after a specified interval, called the *short inter-frame space* (SIFS), that is less than DIFS. If an acknowledgment is not received, the data frame is presumed to be lost, and a retransmission is scheduled. The value of CW is set to CW_{min} in the first transmission attempt, and is doubled at each retransmission up to a pre-determined value CW_{max}. Retransmissions for the same data frame can be made up to a pre-determined retry limit, L, times. Beyond that, the pending frame will be dropped [3]. The contention window update can be summarized as follows:

$$CW_{new} = \begin{cases} 2\,CW_{current}\ , \text{ transmission failure- up to max value} \\ CW_{min}\,, \text{ transmission success} \end{cases}$$

The metrics used in comparing protocol performance are 1) the Packet Delivery Fraction (PDF), which represents the ratio of the number of the successfully delivered data packets to their destinations versus the number of all data packets being sent; 2) the average end to end delay, which measures the average required time in seconds to receive a packet; and 3) the throughput, which is the amount of data successfully transferred through the channel in a given time period. It is measured in kilo bits per second (kbps) [15].

3. History Based Adaptive Backoff (HBAB) Algorithm

This paper proposes a novel backoff algorithm, the History Based Adaptive Backoff (HBAB) algorithm, in which the history of the past trials for transmission is taken into account. In short, HBAB modifies BEB in such a way that the history of the past trials of transmission is taken into account. In order to do that, HBAB defines three variables: **the contention window size (*CW*)**, which holds the current contention window size, **the multiplicative factor**, **α**, which is used to update *CW* and **the *ChannelState***, which captures a snapshot of the medium (or channel) representing its most recent busy/free states. The first variable is common with BEB whereas the latter two variables are HBAB-specific.

The parameter α is a multiplicative factor used to update CW value. A similar multiplicative factor is implicitly defined in the original IEEE 802.11 BEB to have the value of two since CW is doubled upon each transmission failure, i.e. the current CW is multiplied by two to get the new CW. However, α is different from that implicit definition in two aspects:

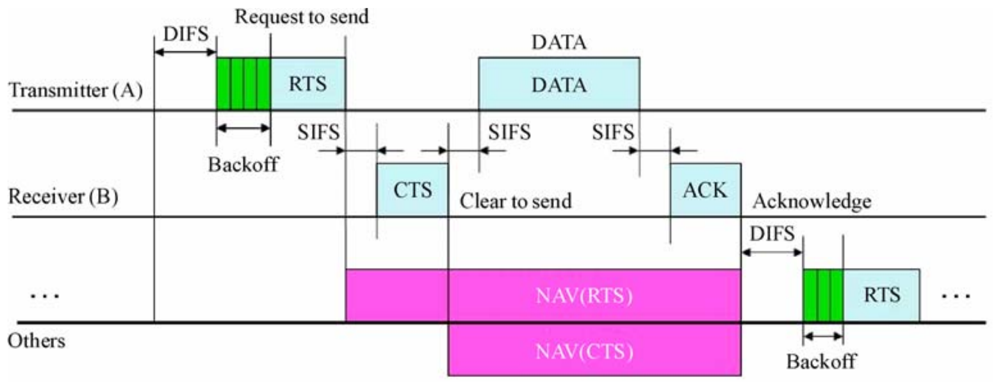


Figure 1. IEEE 802.11 DCF operation.

- Value: theoretically, any positive value greater than 1 can be assigned to α while the multiplicative factor of the original algorithm is fixed at two. The value of α can be either assigned statically, i.e. before runtime and remains constant during runtime or dynamically, i.e. to change according to certain parameters that are sampled during runtime. In this work, only static assignment is implemented.
- Usage: α is used to update CW upon both transmission failures and successes. BEB uses the multiplicative factor only in transmission failure updates since it resets CW upon a transmission success.

In addition to α, HBAB defines another parameter, *ChannelState*, which reflects the most recent history of the medium in terms of its busy/free states. That is, *ChannelState* stores the most recent N states of the medium sensed upon each transmission trial. A free channel state means that the channel is available and no stations are currently transmitting; it is represented by 1. A busy channel state means that there is at least one stations currently transmitting; it is represented by 0. Hence, if N=2 (which is the case in this work), a *ChannelState* of 01 means that the medium had been busy then becomes free. N indicates the depth of this history; the greater N, the deeper captured history of the channel. An obvious tradeoff will be the depth of the channel history versus the available memory to store the states.

The *ChannelState* is updated upon each transmission trial, i.e. each time the station senses the medium trying to transmit a packet. To make room and store the new channel state, the oldest channel state is removed and the remaining stored states are shifted to the left.

When HBAB begins, the three variables, *CW*, α and *ChannelState*, are initialized as follows:

$$CW = CW_{min} \quad \textbf{(1)}$$

$$\alpha = f,\ f > 1 \quad \textbf{(2)}$$

$$ChannelState = 11 \quad \textbf{(3)}$$

where CW_{min} is the minimum value of *CW*, *f* is the chosen value for the multiplicative factor α and *ChannelState* is expressed in binary representation. *f* could be theoretically any positive value greater than one to assure the multiplicative impact on *CW*; in the special case of *f* is exactly one, no updates will take place as the process of updating *CW* involves multiplication or division as will be shown later in this section. If *f* is a positive value less than one, then the multiplicative impact is reversed: multiplying by α decreases the multiplicand whereas dividing by α increases it. Negative values of *f* are meaningless since α is used to update a counter.

If the current transmission trial has failed, *CW* is updated to be the current value of *CW* multiplied by the multiplicative factor α. Thus, the station would wait for a number of time slots equal to the new value of *CW* before attempting transmission again. If, however, the current transmission trial has succeeded, further checks need to be done in order to calculate the updated, namely:

1) If the value of the *ChannelState* represents two consecutive busy states, the new value of *CW* would be the current *CW* divided by α.

2) Otherwise, *CW* is reset to CW_{min}.

Equation (4) illustrates the possible update schemes:

$$\mathrm{CW} = \begin{cases} \mathrm{CW} \cdot \alpha, \mathit{transmission\ \ failure} \\ \mathrm{CWmin,\ transmission\ success,\ ChannelState} \neq 00 \\ \dfrac{\mathrm{CW}}{\alpha},\ \mathrm{transmission\ success,\ ChannelState} = 00 \end{cases} \quad \textbf{(4)}$$

where ChannelState is expressed in binary representation.

Table 1 shows the suggested CW updating method per state check (0 indicates a busy channel and 1 indicates a free channel):

Compared to BEB adopts a similar incrementing approach. While BEB doubles the value of *CW* upon a transmission failure, HBAB multiplies the value of *CW* by α. As for the decrementing approach, it is somehow different. Instead of resetting *CW* to CW_{min} all the time as in BEB, HBAB first checks the value of the *ChannelState* is not busy for two consecutive times provided that the current channel state is free (the latter condition is indicated by the transmission success). In case the channel state is busy for two consecutive times (which indicates a congested medium), *CW* is divided by α instead of resetting to CW_{min}. In other words, a slow decrease in

Table 1. Possible CW update values based on HBAB operation.

Current state	State	CW value
0 (busy)	00(busy, busy) 01 (busy, free) 10 (free, busy) 11 (free, free)	CW = CW *(α)
1 (free)	11 (free, free) 01 (busy, free) 10 (free, busy)	$CW = CW_{min}$
	00(busy, busy)	CW = CW / (α)

CW is preferred in this case since a congested medium (for at least two states) is unlikely to be free upon a single success.

As for the overhead that comes with implementing HBAB, it is due to two factors: extra memory space for new variables and extra computations for additional operations. As for memory, the extra storage space is needed for the new variables: ChannelState and eight other utility variables. As for computations, five additional operations are needed to determine the next value of CW, which are multiplication, shifting, 'if' conditional statement, memory read and memory write operations (to update the hardware registers). An implicit power overhead is associated with the additional operations.

4. Simulation Results

This section shows the performance of HBAB-based IEEE 802.11 against the BEB-based IEEE 802.11 under different network loads. The simulation is carried out in QualNet network simulator version 4.0 [17]. Each point on every graph represents the average of 10 trials using different SEED values to minimize the effect of outliers. The SEED value is used to initialize the random number generator, which is used for node placement and mobility among other usages. Each SEED was used twice: once for an IEEE 802.11 simulation run and the other for the corresponding HBAB simulation run under the same parameters. That ensures the testing has been conducted under the exact conditions, including any random-based parameter.

4.1. Simulation Environment

Our simulation modelled a network of 50 mobile nodes placed randomly within a 1000 × 1000 meters area. Radio propagation range for each node is 250 meters and channel capacity is 2 Mbits/sec. As for node mobility, the random waypoint model is used. In this model, a node chooses a random point in the network, and moves towards that point at a constant speed. The speeds are uniformly chosen between the minimum and maximum speeds set to 0 m/s and 10 m/s, respectively. When the node reaches its destination, it stays there for a certain pause time (fixed to be 20 seconds in this paper), after which it chooses another random destination point and repeats the process.

All simulations last for 600 seconds. The data traffic is generated by Constant Bit Rate (CBR) sessions initiated between random source and destination pairs. Each session lasts until the end of the simulation. Table 2 shows the common simulation parameters.

Table 2. Simulation parameters.

Parameter	Value
Area	1000×1000m
Number of nodes	50
Simulation time	600s
Packet size	512 bytes
Packet rate (per connection)	4 packets/s
Mobility pattern	Random waypoint with max speed of 10m/s and pause time of 20s
Traffic type	Constant bit rate (CBR)
CW_{min}, CW_{max}	31, 1023

Simulation results are shown in Table 3 and Figures 2 through 4. Table 3 shows a detailed percentage improvement of HBAB averaged for all connections loads per values of α (compared to IEEE 802.11). Figures (2)–(4) show the individual PDF, throughput and average delay comparisons between HBAB (for selected values of α) and IEEE 802.11. It was noted that for all values of α and under most of the tested networks loads, HBAB had outperformed IEEE 802.11 for all of the three performance metrics. The only exception was the lowest network load (the ten connections) for higher values of α. In general, performance improvement tended to be higher for smaller values of α; in other words, as the value of α increases, the percentage improvement in HBAB performance compared to IEEE 802.11 decreases.

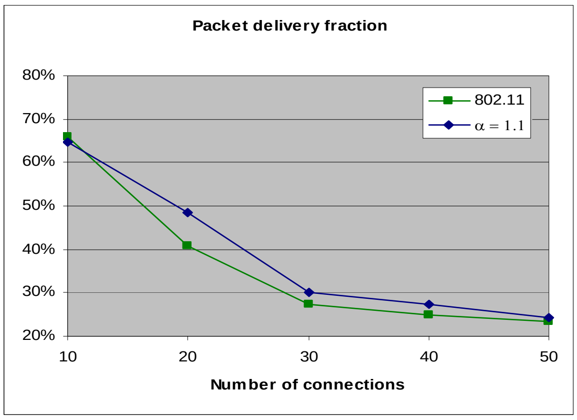
As for each of the performance metrics, the PDF maintained a descending pattern of improvement as the value of α increased (with the highest improvement at α = 1.1), while the average delay showed a different behavior; it reached the peak at α = 1.2 and recorded another increase at α = 1.4. Throughput showed similar behavior to PDF. Overall, the best performance improvement was obtained for α = 1.2 (with 33.51% improvement in delay and 7.36% improvement in PDF compared to IEEE 802.11).

Table 3. Performance improvement of HBAB.

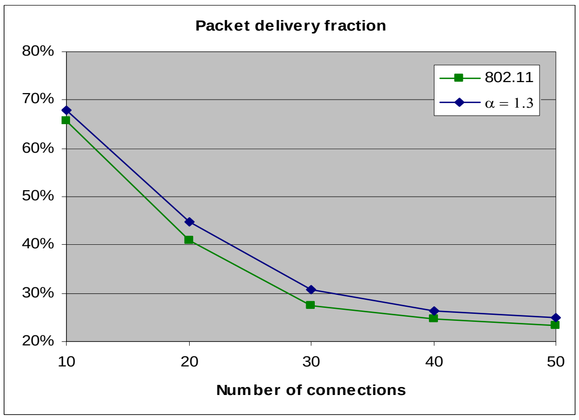
α value	Improvement (compared to IEEE802.11)		
	PDF	Average delay	Throughput
1.1	8.60%	12.13%	8.56%
1.2	7.36%	33.51%	7.40%
1.3	7.70%	14.43%	7.57%
1.4	5.73%	17.48%	5.71%
1.5	5.33%	9.28%	5.32%
1.6	5.04%	6.95%	4.93%
1.7	5.69%	3.38%	5.68%
1.8	5.87%	0.89%	5.89%
1.9	5.36%	1.33%	5.46%
2	3.59%	-16.39%	6.46%

5. Linux Testbed Results

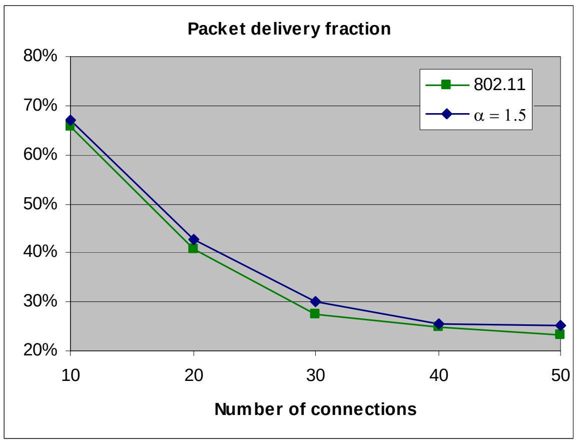
The testbed that was used to test HBAB for real time performance comprises 4 Linux-based stations: 3 desktops and 1 laptop configured to work in the ad-hoc mode. The backoff algorithm itself was loaded on an Atheros-based wireless adapter [18] running the Madwifi driver [19]. All of the four stations ran Fedora 7 distribution [20] and the Linux kernel version 2.6.21. Although Atheros chipset and Madwifi driver offer a level of coding flexibility not present in other chipsets, implementing the backoff algorithm fully was not possible because packet



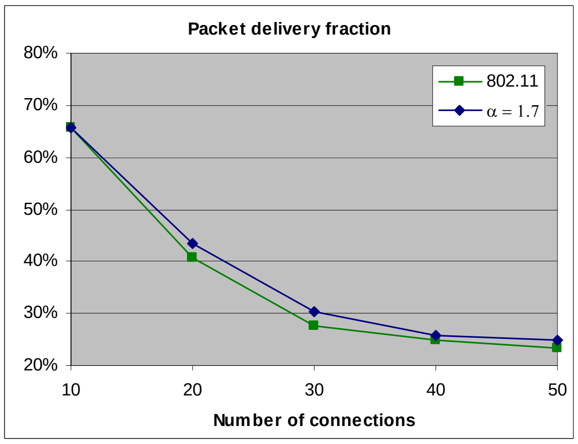
(a)



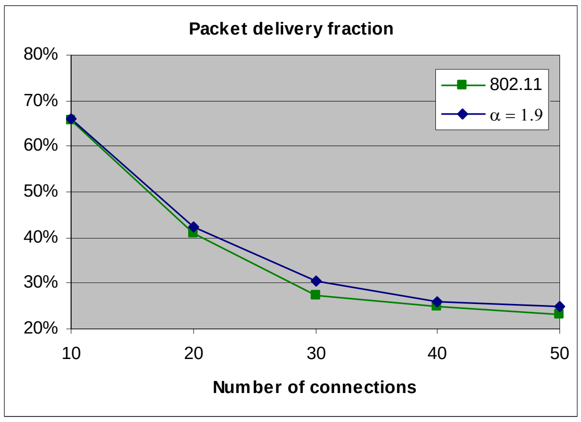
(b)



(c)



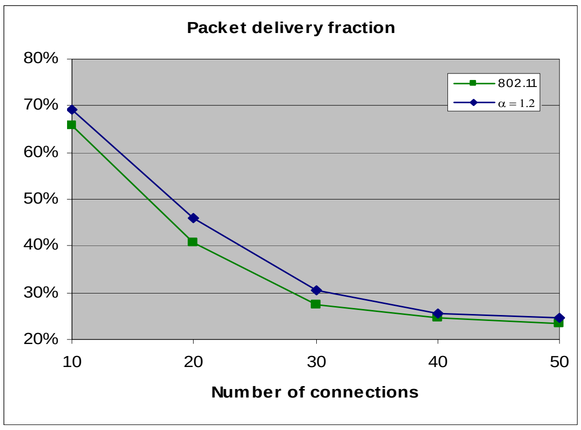
(d)



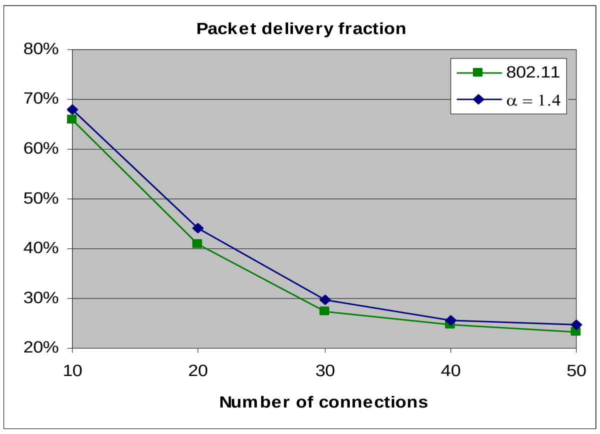
(e)

Figure 1. PDF of HBAB (α= 1.1, 1.3, 1.5, 1.7 and 1.9) vs BEB-simulation.

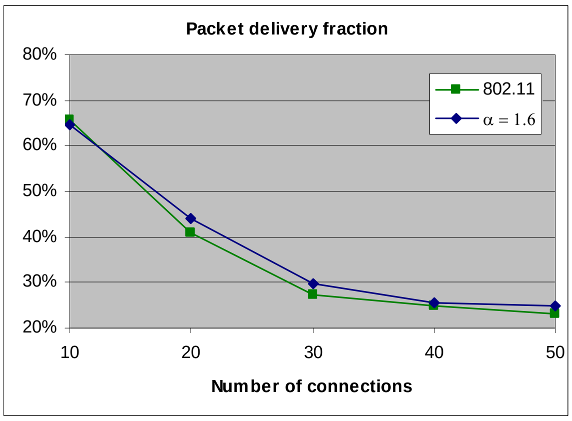
transmission trial handling is done per-packet, not per transmission. That is, a packet success in Madwifi means that a packet has been delivered to the destination regardless the number of retransmissions it had encountered. Similarly, a packet failure means that the packet had failed all the possible transmission trials (until hit-ting the maximum retry limit). Not being able to access the packets per retransmission means that implementing a backoff algorithm per transmission is not possible. Alternatively, a per-packet variant of HBAB has been pro-



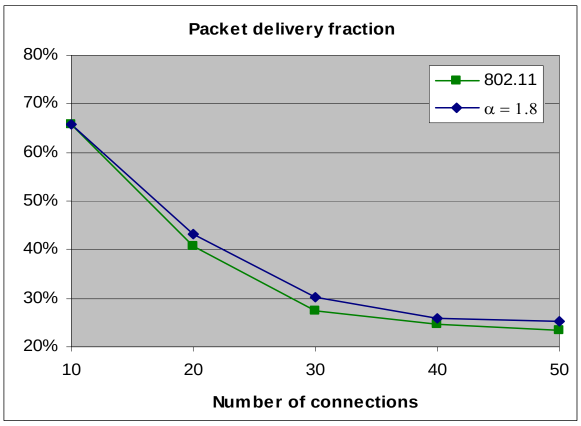
(a)



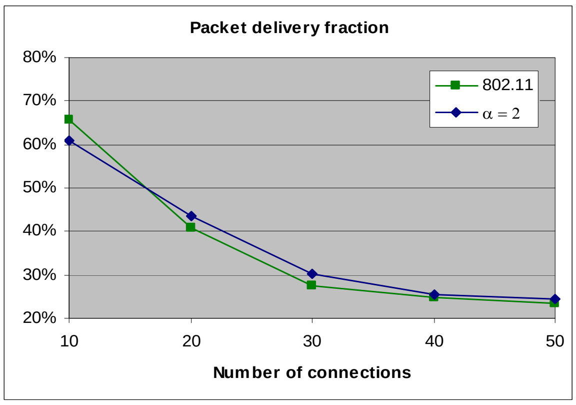
(b)



(c)



(d)

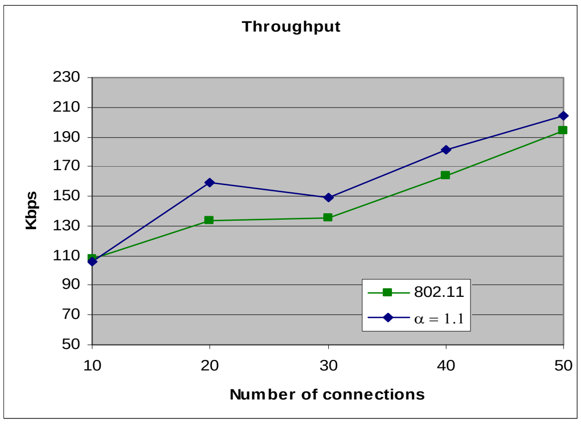


(e)

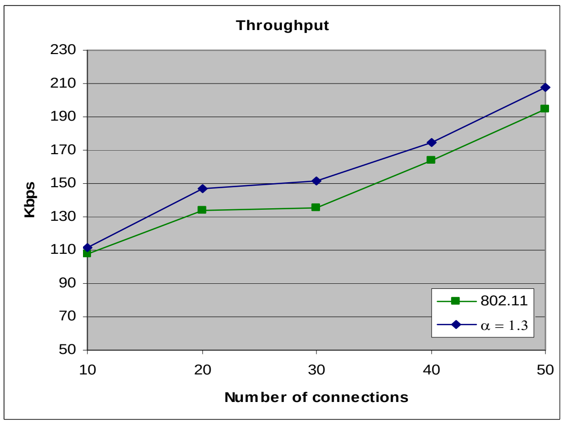
Figure 2. PDF of HBAB (α = 1.2, 1.4, 1.6, 1.8 and 2) vs BEB-simulation.

posed to demonstrate the new backoff algorithm. Table 4 shows the wireless adapters used in this work along with their specifications.

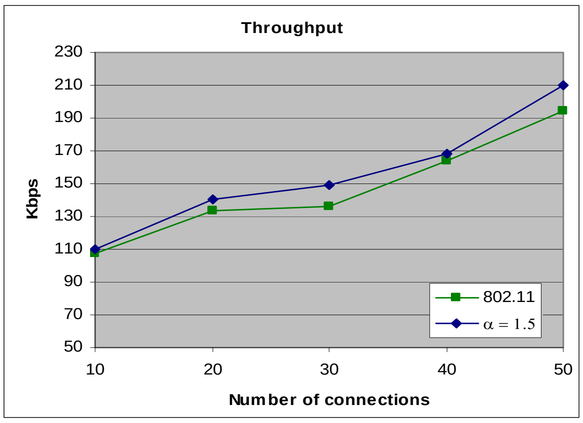
A number of Linux-compatible software tools were used to implement and evaluate HBAB. The network load traffic was generated using the Multi-Generator (MGEN) tool, which is open source software that provides the ability to perform IP network performance tests and measurements using UDP/IP traffic [21]. The TRace Plot Real-time (TRPR) software tool was used to analyze the output files generated by MGEN and calculate the performance metrics used in this work [22]. In order to



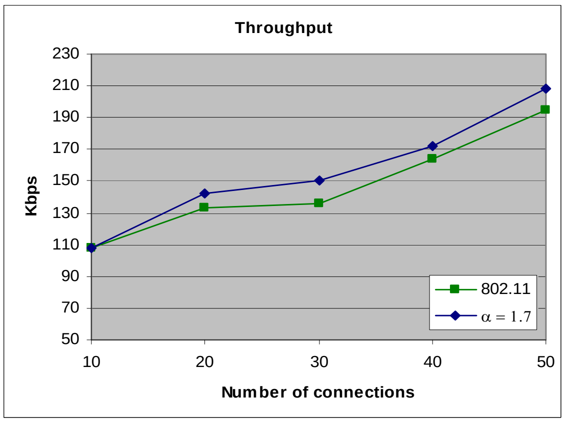
(a)



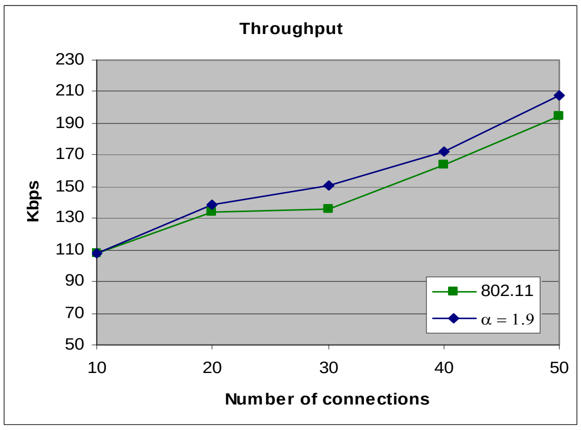
(b)



(c)



(d)



(e)

Figure 3A. Throughput of HBAB (α = 1.1, 1.3, 1.5, 1.7 and 1.9) vs BEB-simulation.

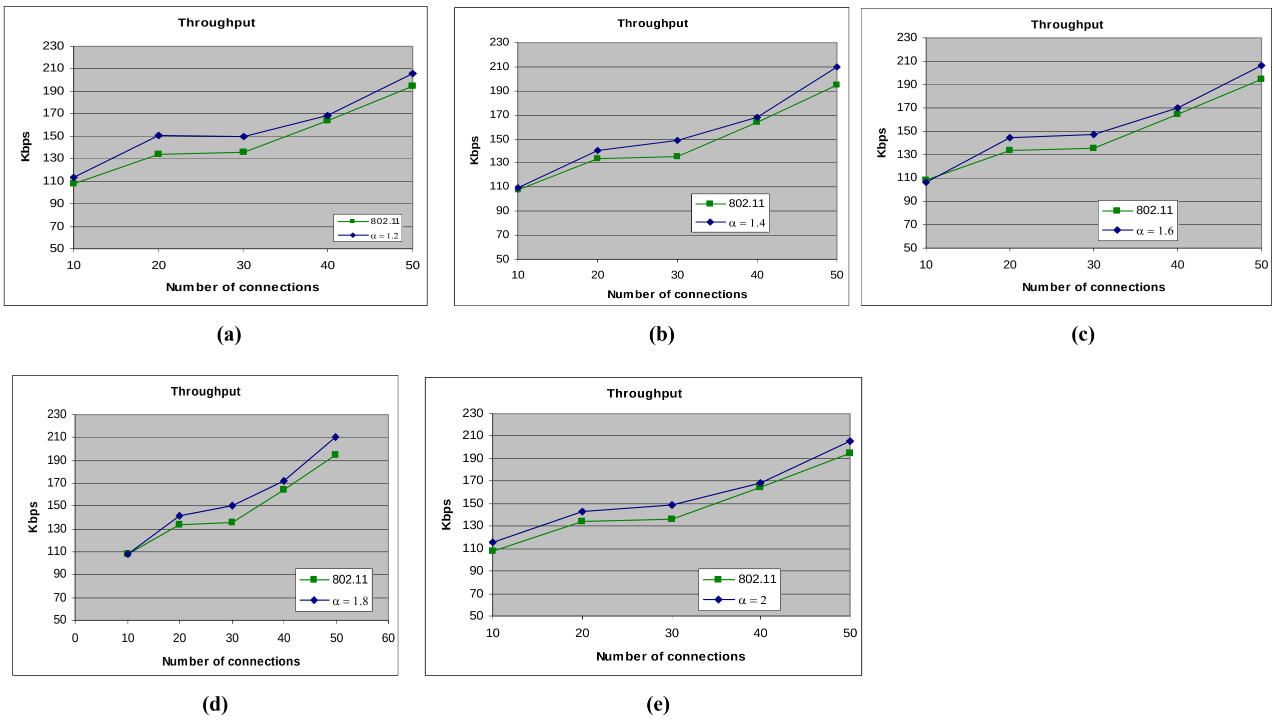


Figure 3B. Throughput of HBAB (α = 1.2, 1.4, 1.6 and 1.8) vs BEB-simulation.

monitor the network and capture live packets as experiments were conducted, Wireshark network protocol analyzer was used [23]. Forming a MANET needed a suitable routing protocol, which was the DSR routing protocol in this work (DSR-UU implementation) [24].

In order to evaluate HBAB real-time performance, a number of testing scenarios were conducted in both simulation and Linux testbed environments. In all testing scenarios, the main connection that was used to evaluate HBAB performance, i.e. to generate the fore ground traffic, was established between Station 2 (source; HBAB-loaded) and Station 4 (destination). The packet generation rate was changed per testing scenario. Other connections were established between Stations 1 and 3 as well as Stations 2 and 4 to generate background traffic, whose packet generation rate also varied per testing scenario. The performance measures (average delay, loss fraction and throughput) were extracted from the log files produced at Station 4 after each testing scenario. Traffic generation rate was changed from 100 to 200 packets/s per station in steps of 20. All testbed scenarios were conducted using the special per-packet variant of HBAB (with varying the multiplicative factor α from 1.1 to 2.0 in steps of 0.1). Table 5 further shows the parameters of the testing scenarios.

The results of the Linux testbed experiments are shown in Table 6 and Figures (5)–(7). Table 6 shows a detailed percentage improvement of HBAB averaged for all connections loads per values of α (compared to IEEE 802.11), whereas Figures (6)–(8) show the individual PDF, throughput and average delay comparisons between HBAB (for selected values of α) and IEEE 802.11. Since HBAB implementation in the Linux testbed was a bit different from simulation, performance results were different as well. In general, there was no consistent performance improvement pattern in the testbed experiment results as illustrated in simulation. HBAB outperformed IEEE 802.11 for almost all values of α (except α = 1.1) with respect to the average delay, and for half of the values of α with respect to the PDF and throughput. Overall, the best performance improvement was obtained for α = 1.7 (with 34.30% improvement in delay and 1.40% improvement in PDF compared to IEEE 802.11).

6. Conclusions

A MANET is an infrastructure-less network connecting a number of mobile nodes via wireless media. The special characteristics of MANETs, such as mobility and absence of centralized control, have made the provisioning of end-to-end QoS guarantees a very challenging problem.

In this work, an adaptive backoff algorithm was developed based on the IEEE 802.11 MAC to provide better QoS performance (especially delay) in MANETs. In

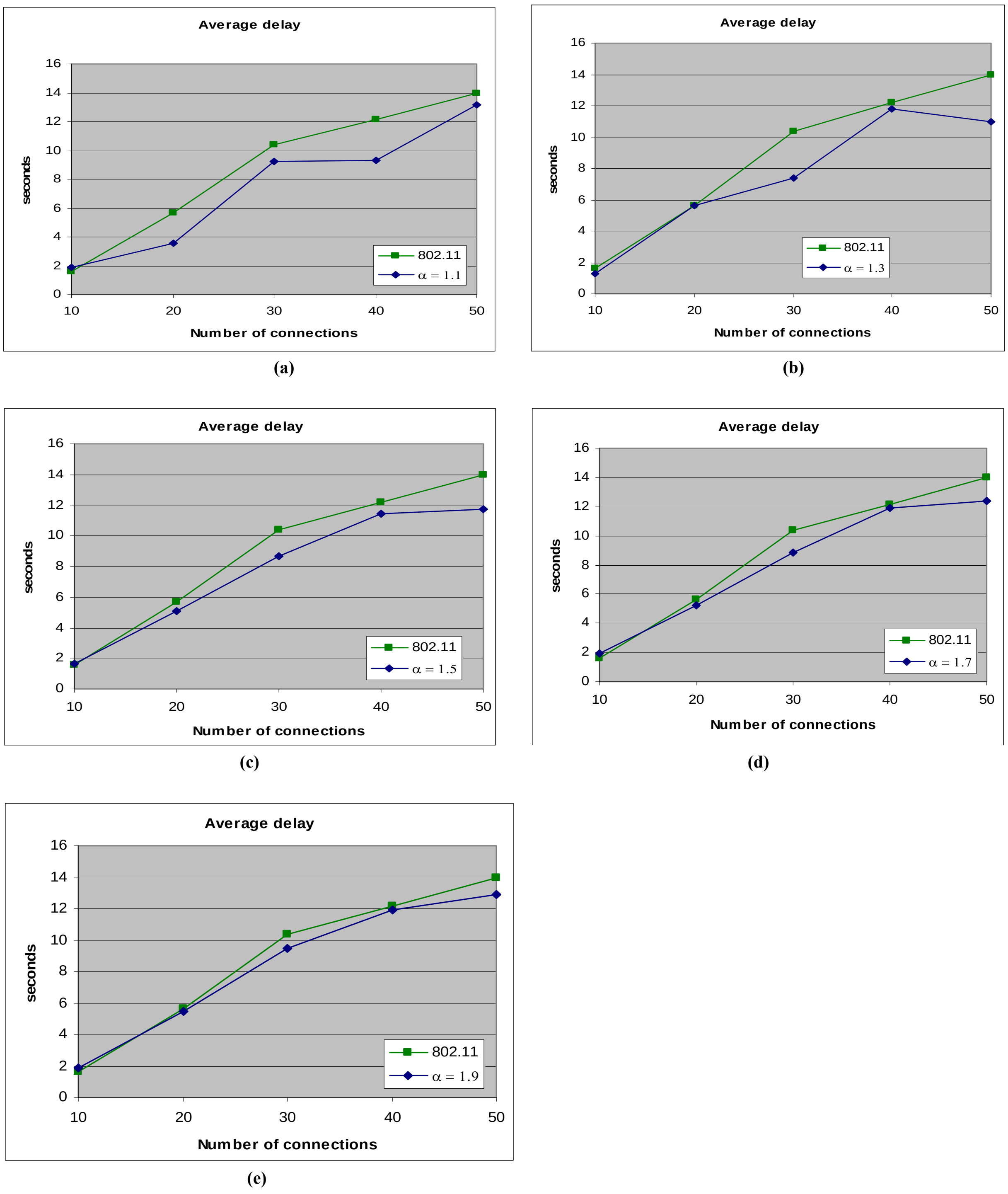
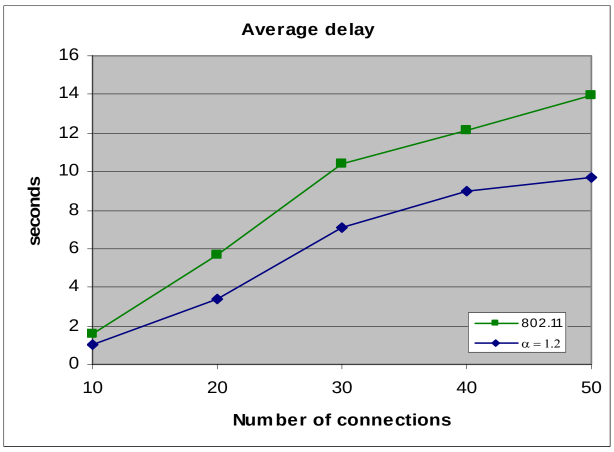


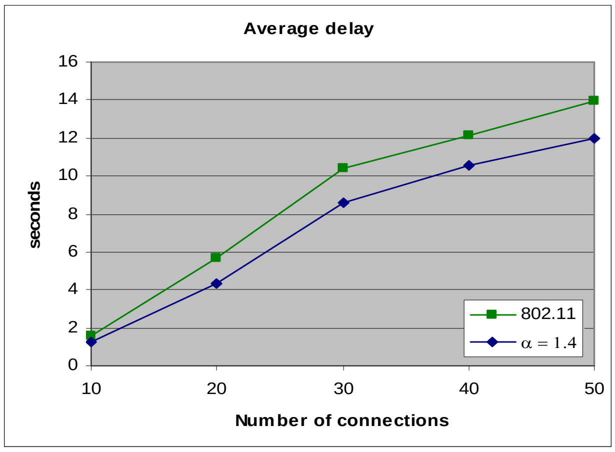
Figure 4A. Average delay of HBAB (α = 1.1, 1.3, 1.5, 1.7 and 1.9) vs BEB-simulation.

particular, the IEEE 802.11's Binary Exponential Back off (BEB) algorithm was replaced with a History-Based Adaptive Backoff (HBAB) algorithm that updated the value of the BEB's Contention Window (CW) according to the channel state over a period of time. In addition to the channel state, HBAB utilized a multiplicative factor, α, to increase or decrease CW value.

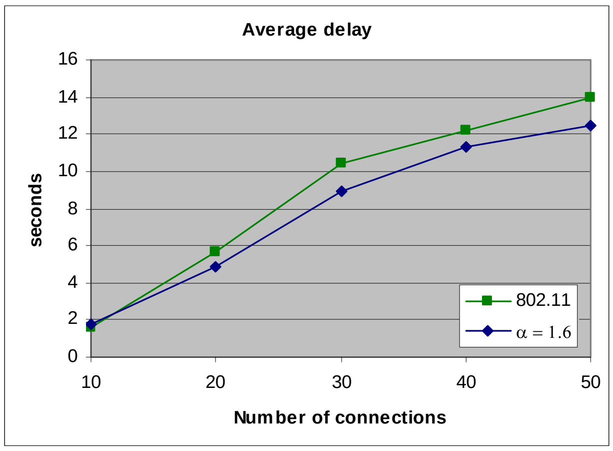
HBAB was tested by simulation (using QualNet simulation package) and implemented in a Linux-based testbed. The simulation environment featured fifty nodes moving in random way-point fashion within 1000 square meters area, whereas the Linux testbed comprises four



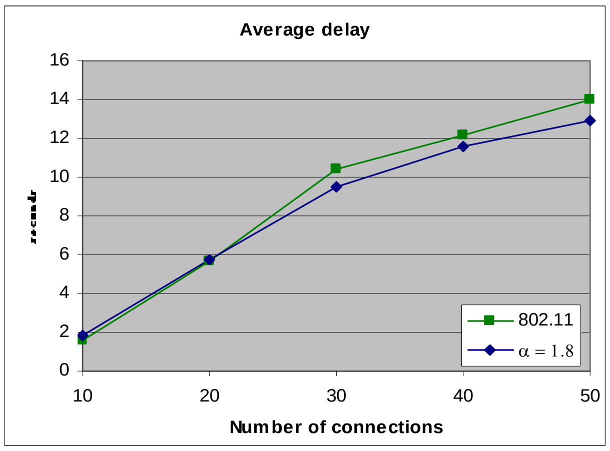
(a)



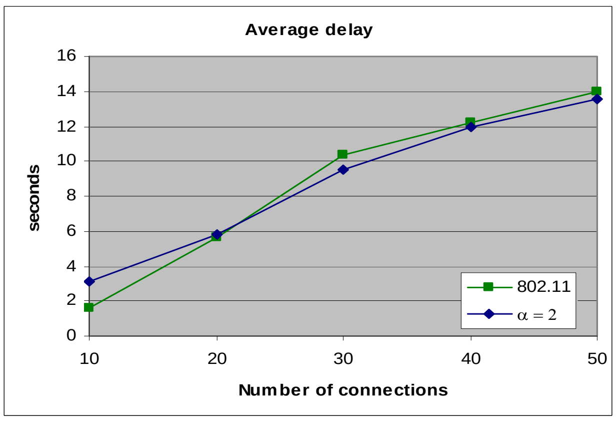
(b)



(c)



(d)



(e)

Figure 4B. Average delay of HBAB (α = 1.2, 1.4, 1.6, 1.8 and 2) vs BEB-simulation.

Table 4. Wireless cards specifications.

Make and model	Dlink DWL-122	Netgear WPN-511	Dlink DWA-110
Chipset	Intersil Prism II	Atheros AR5002 (AR5212)	Ralink RT2501U
Network standards	802.11 b	802.11 b and g	802.11 b and g
Host interface	USB	PCMCIA	USB
Wireless adapter driver	Linux-wlan-ng 0.2.8	Madwifi 0.9.3.1	rt73 1.1.0.0

Table 5. Testbed parameters.

Parameter	Value
Area	4×4m
Number of stations	4
Testing scenario duration	Average of 2 trials, each of 60 s
Packet size (foreground and background traffic)	512 bytes
Packet rate (foreground and background traffic)	100, 120, 140, 160, 180, 200 packets/s (per station)
Background traffic type	UDP Constant Bit Rate (CBR)

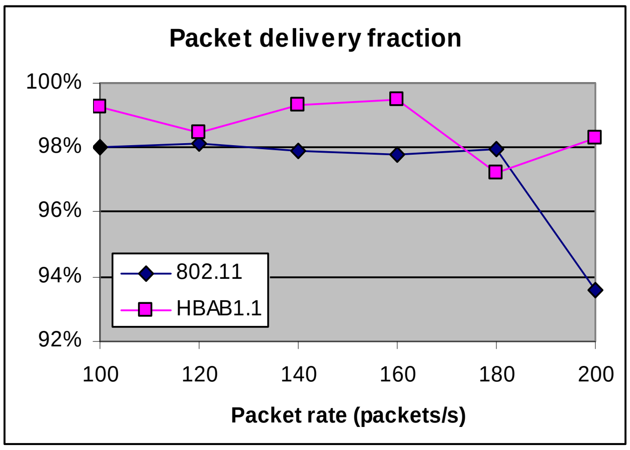
Table 6. HBAB performance improvement (compared to IEEE 802.11) for the hardware testbed experiments.

α value	Improvement (compared to IEEE 802.11)		
	PDF	Average delay	Throughput
1.1	1.50%	-0.51%	1.50%
1.2	-1.49%	6.10%	-1.49%
1.3	0.74%	1.11%	0.74%
1.4	0.72%	13.11%	0.73%
1.5	0.33%	14.75%	0.32%
1.6	-2.47%	10.95%	-2.48%
1.7	1.40%	34.30%	1.40%
1.8	0.29%	13.37%	0.29%
1.9	-1.37%	17.57%	-1.37%
2	-3.75%	4.35%	-3.75%

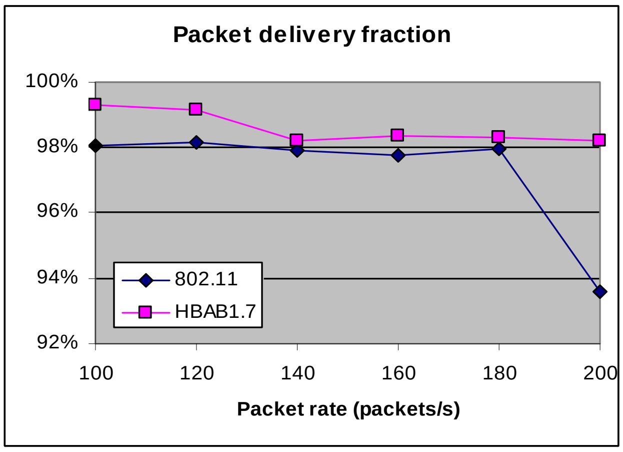
stationary nodes equipped with the necessary hardware and software to form a MANET. Due to some hardware limitations, it was not possible to implement HBAB in the Linux-based testbed with its original design as im ple-mented in simulation. Therefore, a special variant of HBAB was developed for that purpose as detailed in Section 5.

Simulation results have shown significant improvement in IEEE 802.11 performance measures upon using HBAB over BEB. For HBAB ($\alpha = 1.2$), the average improvements in PDF and average delay were 7.36% and 33.51%, respectively. The Linux-based testbed results for $\alpha = 1.7$ showed 1.40% and 34.30% improvement in PDF and average delay, respectively. Based on those results, it could be concluded that the lower values of α (which are associated with lower delay) combined with HBAB's novel approach in updating CW provided better channel utilization and eliminated unnecessary waiting time. That was revealed as a decrease in the average delay and increase in PDF and throughput. The overhead associated with implementing HBAB is due to two factors: extra memory space for nine new variables and extra computations for five additional operations.

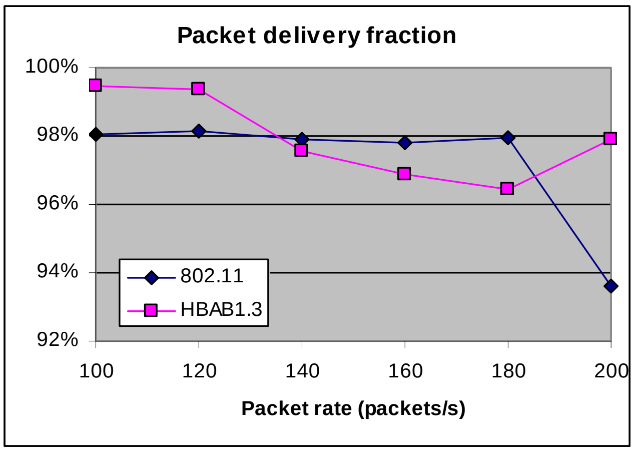
There are a number of open issues for future development. For instance, HBAB can be applied to different Access Categories (ACs) in IEEE 802.11e instead of only one category as in the legacy IEEE 802.11. Also, the multiplicative factor α can be implemented in such a way that it can be dynamically modified according to the network load and/or medium status.



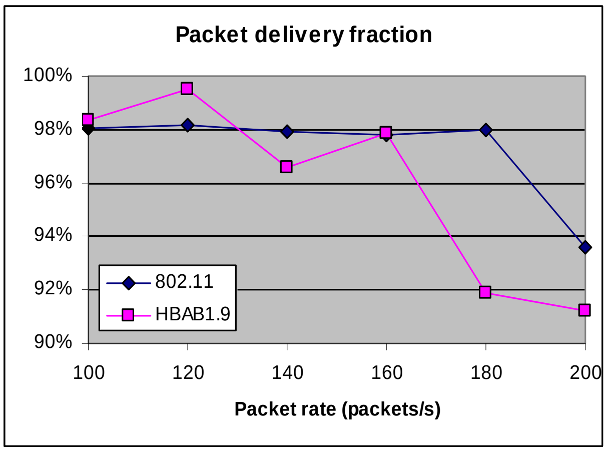
(a)



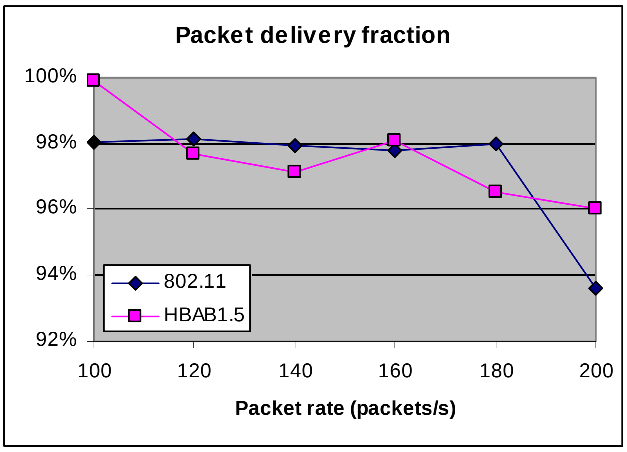
(b)



(c)

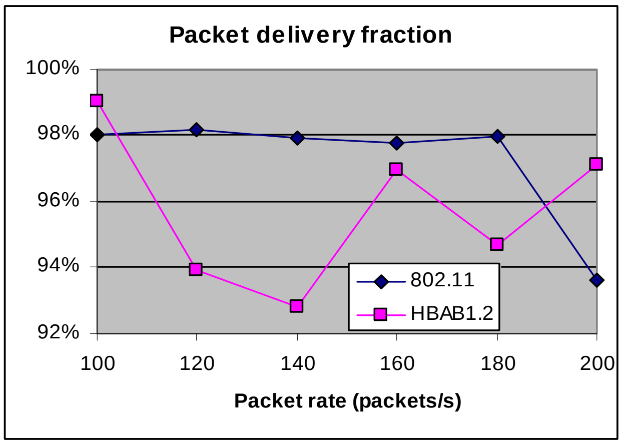


(d)

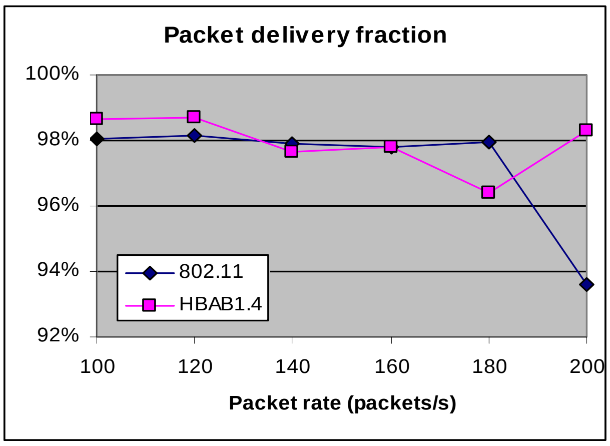


(e)

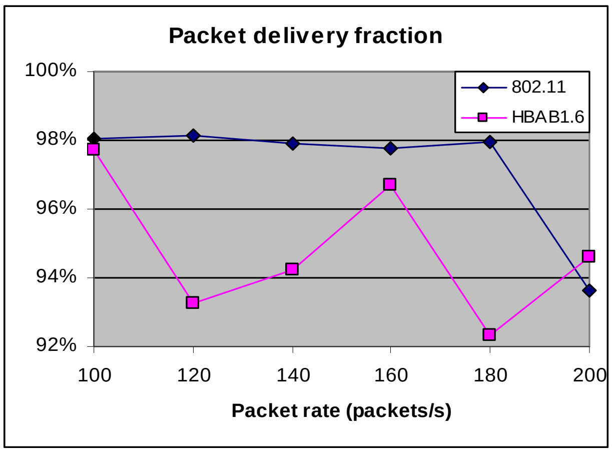
Figure 5A. PDF of HBAB (α = 1.1, 1.3, 1.5, 1.7 and 1.9) vs BEB-Linux implementation.



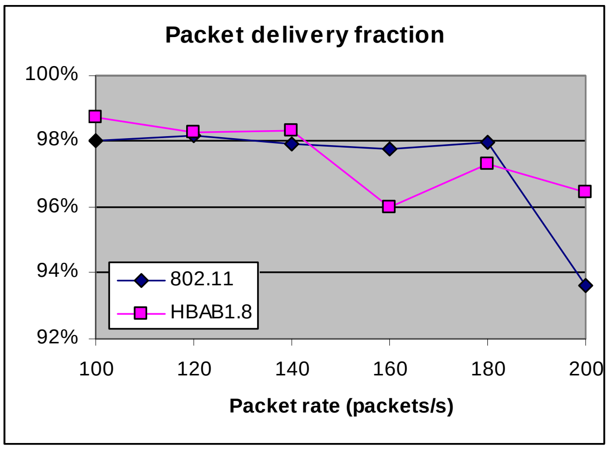
(a)



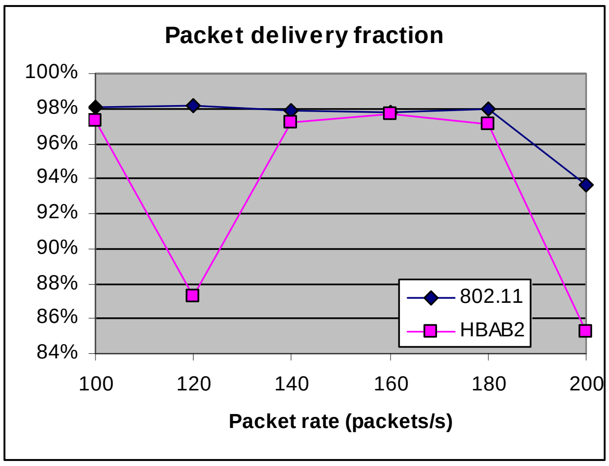
(b)



(c)

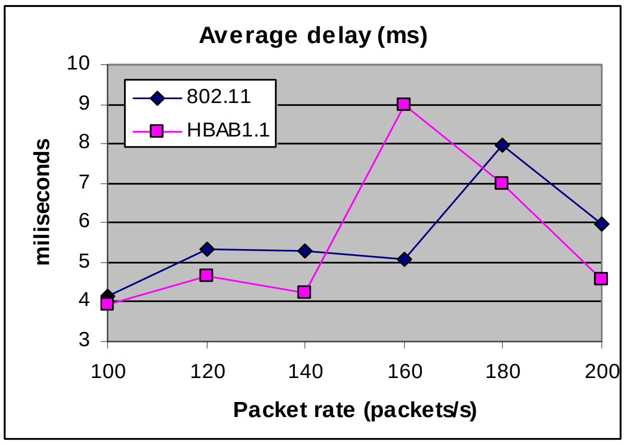


(d)

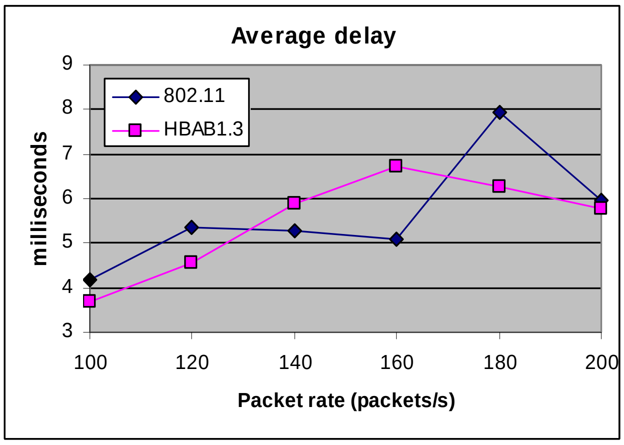


(e)

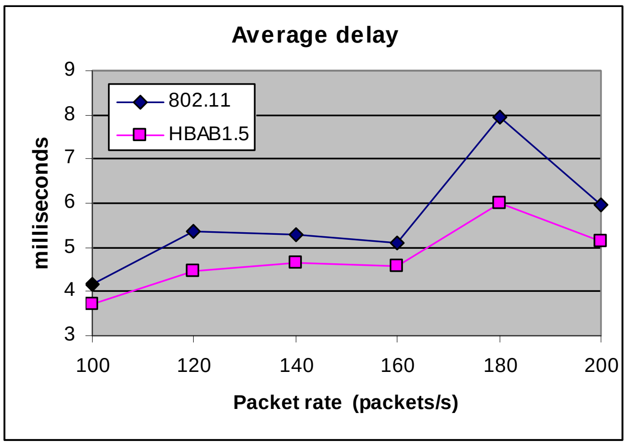
Figure 5B. PDF of HBAB (α = 1.2, 1.4, 1.6, 1.8 and 2) vs BEB-Linux implementation.



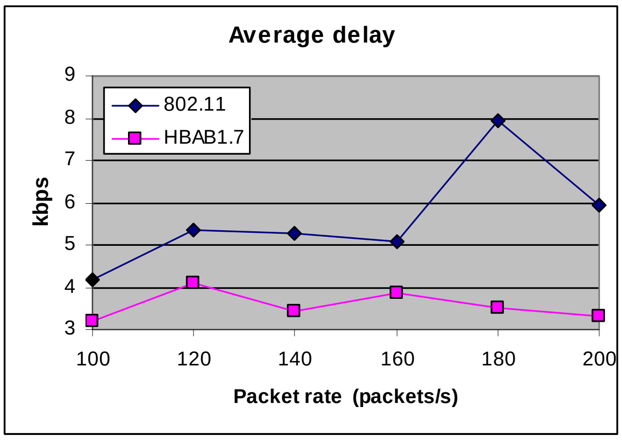
(a)



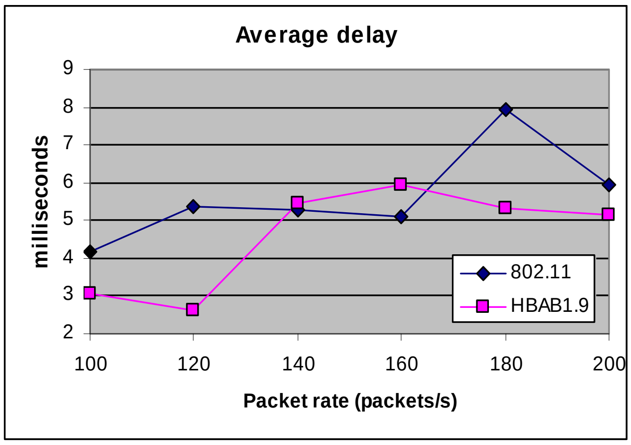
(b)



(c)

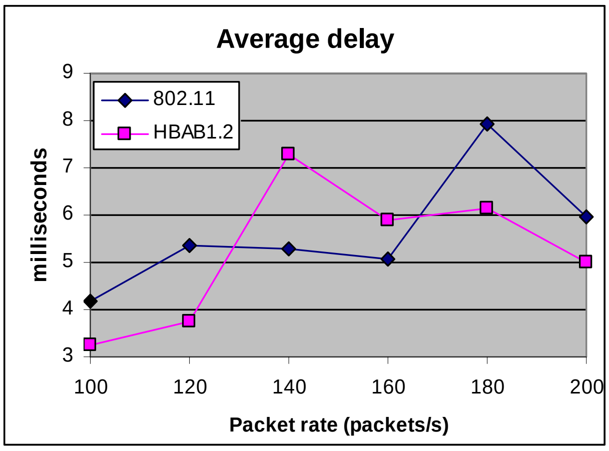


(d)

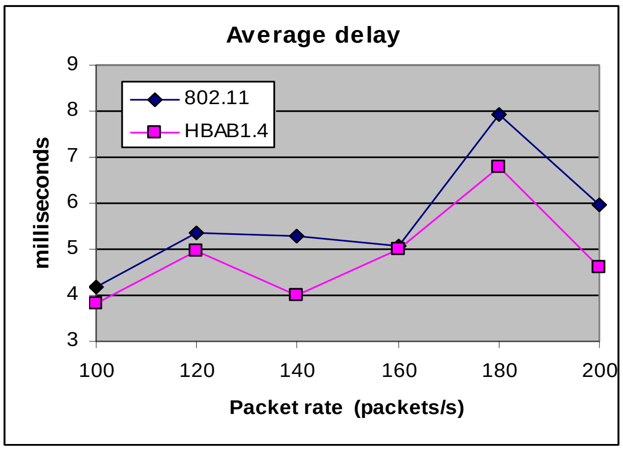


(e)

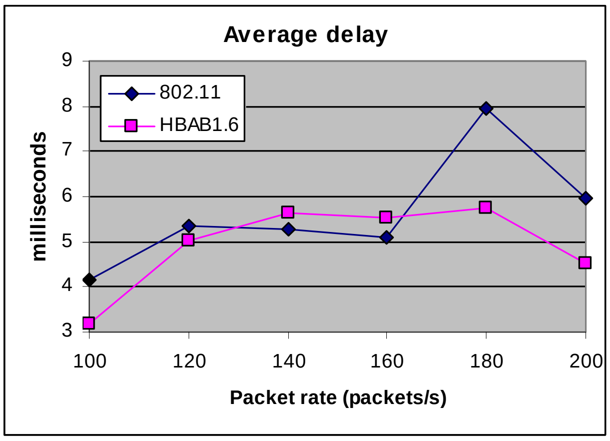
Figure 6A. Average delay of HBAB (α = 1.1, 1.3, 1.5, 1.7 and 1.9) vs BEB-Linux implementation.



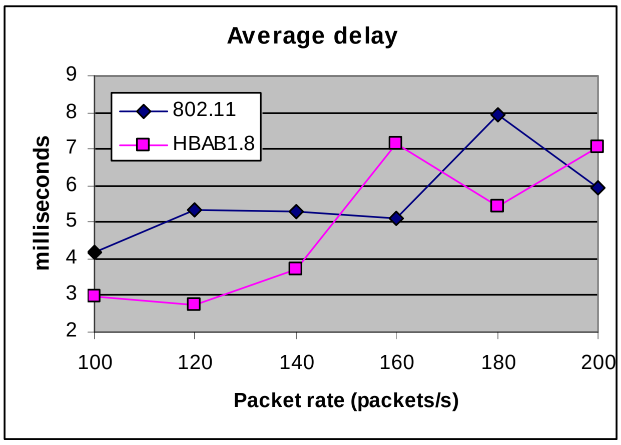
(a)



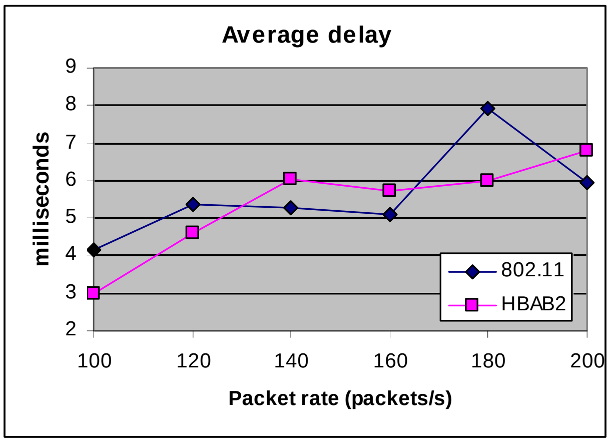
(b)



(c)

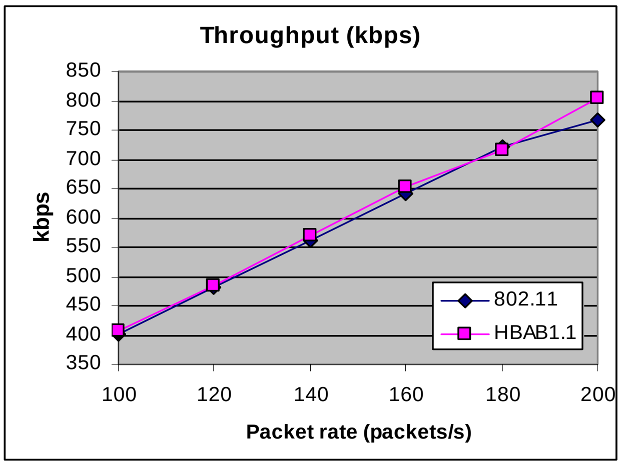


(d)

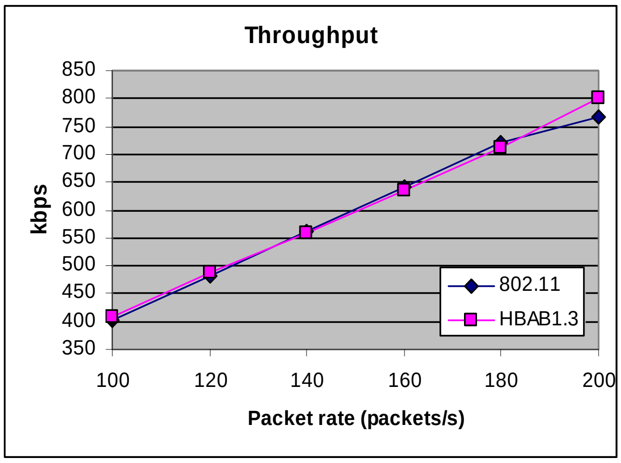


(e)

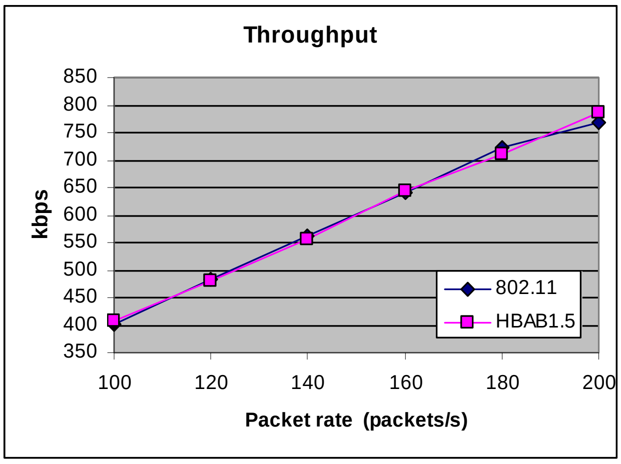
Figure 6B. Average delay of HBAB (α = 1.2, 1.4, 1.6, 1.8 and 2) vs BEB-Linux implementation.



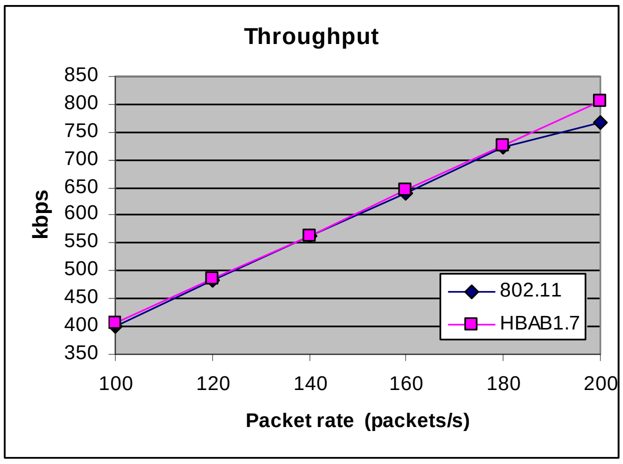
(a)



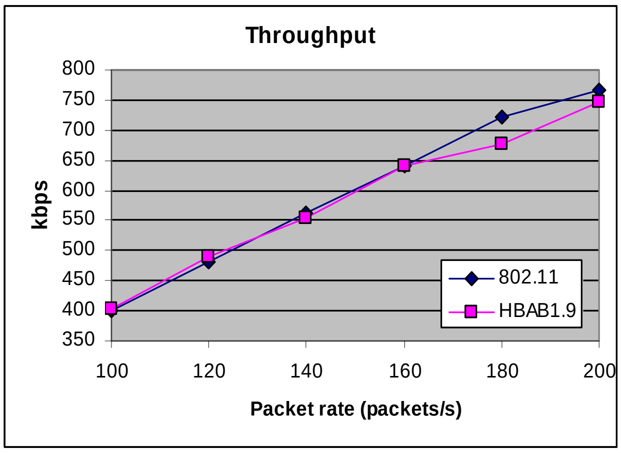
(b)



(c)

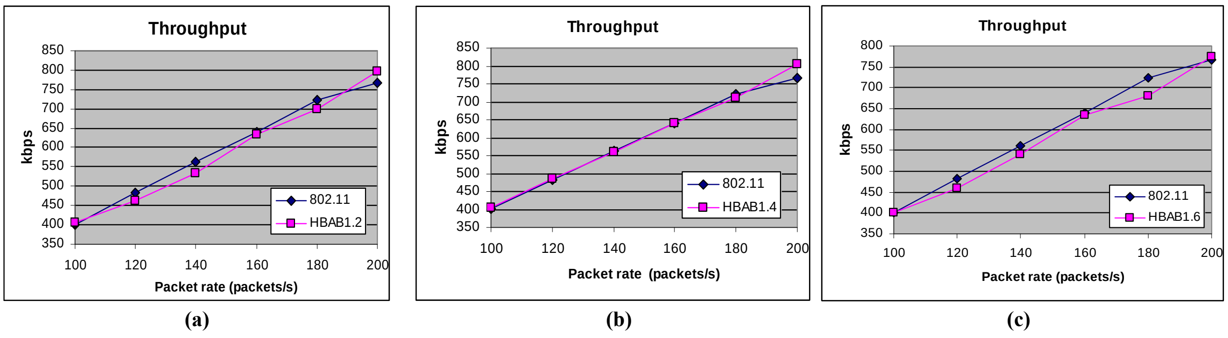


(d)



(e)

Figure 7A. Throughput of HBAB (α = 1.1, 1.3, 1.5, 1.7 and 1.9) vs BEB-Linux implementation.



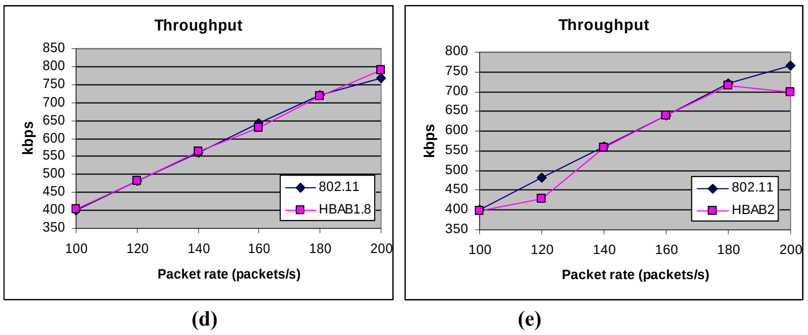


Figure 7B. Throughput of HBAB (α = 1.2, 1.4, 1.6, 1.8 and 2) vs BEB-Linux implementation.

7. References

[1] C. Jones, "A survey of energy efficient network protocols for wireless networks," Wireless Networks, Vol. 7, No. 4, pp. 343–358, 2001.

[2] D. Deng and R. Chang, "A priority scheme for IEEE 802.11 DCF access method," IEICE Transactions on Communications, Vol. E82-B, No. 1, pp. 96–102, 1999.

[3] M. Barry, A. Campbell, and A. Veres, "Distributed control algorithms for service differentiation in wireless packet networks," Proceedings of INFOCOM 2001, Anchorage, AK, USA, April 2001.

[4] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Distributed multi-hop scheduling and medium access with delay and throughput constraints," Proceedings of MobiCOM 2001, Rome, Italy, July 2001.

[5] I. Aad and C. Castelluccia, "Differentiation mechanisms for IEEE 802.11," Proceedings of INFOCOM 2001, Anchorage, AK, USA, April 2001.

[6] Q. Ni, I. Aad, C. Barakat, and T. Turletti, "Modeling and analysis of slow CW decrease for IEEE 802.11 WLAN," Proceedings of PIMRC 2003, Beijing, China, September 2003.

[7] P. Chatzimisios, et al., "A simple and effective backoff scheme for the IEEE 802.11 MAC protocol," Proceedings of CITSA 2005, Orlando, FL, USA, July 2005.

[8] J. A. Moura and R. N. Marinheiro, "MAC approaches for QoS enhancement in wireless LANs," Proceedings of JETC 2005, Lisbona, Portugal, November 2005.

[9] B Li and R. Battiti, "Achieving optimal performance in IEEE 802.11 wireless LANs with the combination of link adaptation and adaptive backoff," Computer Networks Journal, Elsevier Science BV, Vol. 51, No. 6, pp. 1574–1600, 2007.

[10] F. Calì, M. Conti, and E. Gregori, "IEEE 802.11 protocol: Design and performance evaluation of an adaptive backoff mechanism," IEEE Journal on Selected Areas in Communications, Vol. 18, No. 9, pp. 1774–1786, 2000.

[11] R. Bruno, M. Conti, and E. Gregori, "A simple protocol for the dynamic tuning of the backoff mechanism in IEEE 802.11 networks," Computer Networks Journal, Elsevier Science BV, Vol. 37, No. 1, pp. 33–44, 2001.

[12] T. B. Reddy, J. P. John, and C. S. R. Murthy, "Providing MAC QoS for multimedia traffic in 802.11e based multi-hop ad hoc wireless networks," Computer Networks: The International Journal of Computer and Telecommunications Networking, Vol. 51, No. 1, pp. 153–176, 2007.

[13] Q. Nasir and M. Albalt, "History based adaptive backoff (HBAB) IEEE 802.11 MAC protocol," Proceedings of CNSR 2008, Nova Scotia, Canada, May 2008.

[14] Q. Ni, "Performance analysis and enhancements for IEEE 802.11e wireless networks," IEEE Networks, Vol. 19, No. 4, pp. 21–27, 2005.

[15] K. Farkas, D. Budke, B. Plattner, O. Wellnitz, and L. Wolf, "QoS extensions to mobile ad hoc routing supporting real-time applications," Proceedings of AICCSA 2006, Sharjah, UAE, March 2006.

[16] T. Reddy, J. John, and C. Murthy, "Providing MAC QoS for multimedia traffic in 802.11e based multi-hop ad hoc wireless networks," Computer Networks, Vol. 51, No. 1, pp. 153–176, 2007.

[17] Scalable Network Technologies, Inc., QualNet 4.0 product tour, 2006.

[18] Atheros Communications, AR5002 product bulletin, 2007.

[19] Madwifi driver webpage: www.madwifi.org, retrieved on April 15, 2008.

[20] Fedora Linux distribution webpage: www.fedoraproject.org, retrieved on April 15, 2008.

[21] MGEN webpage: http://cs.itd.nrl.navy.mil/work/mgen/, retrieved on April 15, 2008.

[22] TRPR webpage: http://pf.itd.nrl.navy.mil/protools/trpr.html, retrieved on April 15, 2008.

[23] Wireshark webpage: http://www.wireshark.org/, retrieved on August 15, 2008.

[24] DSR-UU webpage: http://core.it.uu.se/core/index.php/DSR-UU, retrieved on August 15, 2008.