TAMPERE UNIVERSITY OF TECHNOLOGY

SONG XING

COMPARISON OF DIFFERENT VERSIONS OF TCP IN 802.11 WLANS

Master of Science Thesis

Examiner: Yevgeni Koucheryavy
Dmitri Moltchanov
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 15 August 2012.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology
XING, SONG: Comparison of different versions of TCP in 802.11 WLANs
Master of Science Thesis, 44 pages, 10 Appendix pages
August 2012
Major subject: Communications Engineering
Examiners: Professor Yevgeni Koucheryavy, Dr. Dmitri Moltchanov
Keywords: Transmission Control Protocol, 802.11, WLANs, acknowledgements, throughputs

Nowadays, Transmission Control Protocol (TCP) is widely used in 802.11 Wireless Local Area Networks (WLANs) because it can provide reliable communication connection between two hosts. This thesis discusses whether a base station in some 802.11 WLAN may be a bottleneck for uplink TCP flows because of delay of acknowledgements (ACKs) in downlink flows when this base station is the only one in the network. The conclusion can be obtained by comparing throughputs of different versions of TCP in 802.11 WLAN. The method is to examine whether there is big difference between the throughputs of different versions of TCP with more or less ACKs. So, the throughputs of normal TCP and TCP with delayed ACKs are compared.

The thesis is composed of six parts. The first part is introduction. The second part is introduction to theoretical background. Some essential background knowledge about TCP and 802.11 protocols are introduced. The third part is problem statement. The question concerned in the thesis is introduced. The fourth part is description of simulations and throughput calculation. The simulations are designed by using the software called Network Simulator 2 (NS2). The throughputs are calculated by using an AWK file. Then, the next part is to show the results, including result figures and discussion. The last part is conclusion.

In the scenarios of this paper, the conclusion indicates that there is no big difference between the throughputs of different versions of TCP with more or less ACKs even though the difference exists statistically. With the number of sources increase, the throughputs of compared versions of TCP decrease and the difference of their throughputs is not as big as some users have expected. So, it is clear that the only base station in some 802.11 WLANs does not become a bottleneck for the uplink TCP flows because of delayed ACKs in the downlink flows. The results of this study indicate that it is not necessary for users to use TCP with less ACKs. In most of cases in daily life, it is enough to use normal version of TCP in 802.11 WLANs which only have one base station.

PREFACE

This Master of Science Thesis, “Comparison of Different Versions of TCP in 802.11 WLANs”, has been carried out in the Department of Communications Engineering at Tampere University of Technology, Finland. This thesis work is to assess whether a base station may be an artificial bottleneck for uplink TCP flows because of the delay of ACKs in the downlink in IEEE 802.11 WLANs. This thesis work is done during the year of 2012 at Networks and Protocol Group, Tampere University of Technology.

When I was working on this thesis, I got a lot of help from my teachers and friends. Firstly, I would like to express my sincere appreciation to my thesis examiner, Professor Yevegeni Koucheryavy, and my supervisor Dmitri Moltchanov. I want to thank Professor Yevegeni Koucheryavy for supporting my thesis work and viewing my paper. I also appreciate the time and effort which are spent on my thesis by Dmitri. He is a really good supervisor and he gave me the greatest help. Secondly, I want to thank Song Tian and Mengnan Wang. They offered their kind help to me when I had difficult problems to finish my thesis. At last, I need to thank my family for supporting me and all other people who helped me.

Tampere, July 2012

Song Xing
song.xing@tut.fi
Insinöörinkatu 60 B 141
33720 Tampere
FINLAND
Tel. +358 417077318

CONTENTS

LIST OF ABBREVIATIONS

ACKs	Acknowledgements
ADDR	Address
AP	Access point
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CWR	Congestion Window Reduced
DCF	Distributed Coordination Function
DIFS	Distributed Inter Frame Space
DSSS	Direct Sequence Spread Spectrum
DST	Destination
ECE	ECN-Echo
FHSS	Frequency Hopping Spread Spectrum
FIN	Finished
GHz	Giga hertz
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISN	Initial Sequence Number
MAC	Media Access Control
Mbit/s	Mega bits per second
MHz	Megahertz
MIMO	Multiple-Input Multiple-Output
NS2	Network Simulator 2
NUM	Number
OFDM	Orthogonal Frequency Division Multiplexing
PHY	Physical
PKT	Packet
PSH	Push
RFCs	Request for Comments
RST	Reset
SEQ	Sequence

SIFS	Short Inter Frame Space
SRC	Source
SYN	Synchronize
TCP	Transmission Control Protocol
URG	Urgent
WLANs	Wireless Local Area Networks

1. INTRODUCTION

Currently, Transmission Control Protocol (TCP) is widely used in IEEE 802.11 Wireless Local Area Networks (WLANs). TCP is popular in networking communication because it can provide reliable connections by using acknowledgments (ACKs). 802.11 protocols were developed by Institute of Electrical and Electronics Engineers (IEEE) for WLANs. 802.11 protocols use Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) in their MAC layer instead of Carrier Sense Multiple Access with Collision Detection (CSMA/CD). CSMA/CA enables all wireless nodes and access points (APs) to fairly share the wireless channels. In current communication networks including 802.11 WLANs which support TCP, there are more than one versions of TCP which are being used. Some people think it is always different for users to use different versions of TCP in 802.11 WLANs. The users have to choose the best version in every scenario. In fact, it is not absolute. The objective of this paper is to verify whether there is always big difference to use different versions of TCP in 802.11 WLANs.

According to the characteristics of TCP and 802.11 protocols, all stations supporting TCP in WLAN including base stations have the same channel access priority. When there is some scenario where uplink TCP flows are sent from multiple wireless nodes and downlink ACKs are sent by only one AP, an interesting question is found: is it possible for a base station being a bottleneck for uplink because of delay of ACKs in downlink? For verifying this problem, some experiments need to be done. And for this, two versions of TCP are chosen for comparing their performance. The second version of TCP can generate less ACKs than the first one because of delay of ACKs. The purpose of experiments is to find out whether there is any obvious difference for throughput of two different versions of TCP which use more or less ACKs. The experiment scenarios are simulated by using the software named Network Simulator 2 (NS2). Then, throughputs of TCP nodes in different scenarios are calculated by analyzing trace files generated from the experiments. Through comparing the difference between the throughputs, it is feasible to find out whether a base station can be a bottleneck for the uplink because of delay of ACKs in the downlink. That is why the topic of this thesis is "comparison of different versions of TCP in 802.11 WLANs".

In this thesis, firstly, some theoretical background related to this thesis needs to be introduced, such as TCP and 802.11. Secondly, it is problem statement. This part is about the problem discussed in this paper. The third part is to introduce the details of simulation scenarios made by using NS2. Then, the next part is to show and analyze the results of the simulations, and the last part is the conclusion.

2. THEORETICAL BACKGROUND

2.1. Transmission control protocol

TCP is one of main protocols compositing of TCP/IP protocol suite. In TCP/IP protocol suite, TCP protocol is responsible for building a connection between two nodes in TCP/IP networks whereas Internet Protocol (IP) protocol is mainly responsible for dealing with data packets [1]. TCP protocols are defined in a series of Request for Comments (RFCs) published by the Internet Engineering Task Force (IETF), especially RFC 793 [2].

In the early days, TCP was developed to provide stable connections for different remote hosts to exchange data reliably in different networks. Soon, the scientists found it was very difficult to transmit the data free of error in heterogeneous networks because each of them has different addressing mode, maximum packet size, delay, and other things. The problems caused that the packets delivered were dropped, damaged, or duplicated. To solve the problems, TCP was designed to detect duplicated packets, check transmission quality by sending ACKs, and retransmit dropped packets. The error detection and retransmission scheme make sure the transmission of data link is maintained. [3]

The good communication network needs reliable connection, and TCP is used widely because of its advantages. TCP is considered as a reliable network communication protocol. Usually, an effective connection between a sender and a receiver is always established before the traffic is generated. The connection is bi-directional, which means it is full-duplex. During the sessions, both senders and receivers keep monitoring the state of the sessions. Each data packets must be acknowledged by the receivers. Moreover, if the receivers do not get the packets, they will inform the senders and the senders will retransmit the packets. So, reliable links are established by applying TCP. [3]

2.2. TCP segment format

In order to analyze throughputs of different versions of TCP, it is necessary to mention TCP segment format. TCP is composed of a series of byte-stream-oriented protocols. TCP deals with data from multiple data streams. It segments the data, and then adds TCP headers to create TCP segments which are often referred to as TCP packets. After this, the TCP segments are encapsulated into Internet Protocol datagram (IP datagram) delivered in TCP/IP networks. [3]

A single TCP segment contains a TCP header and a data section. The TCP segment is composed by multiple overlain 32 bits data segments [3]. Basic structure of a TCP segment is shown as follows:

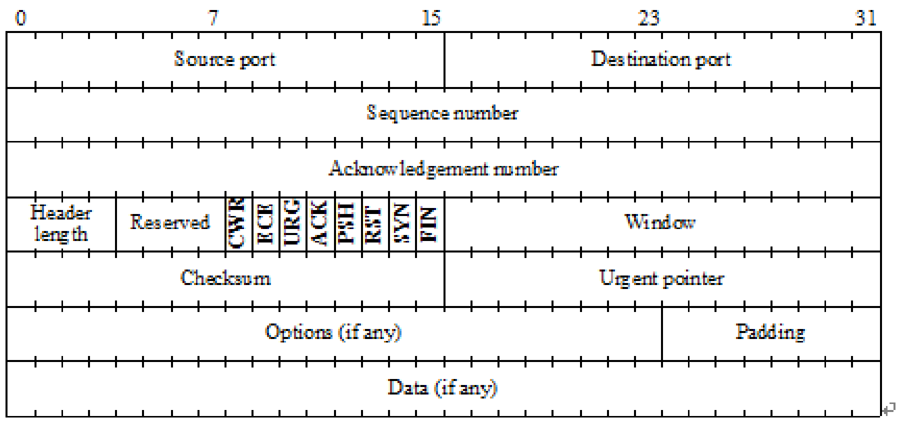


Figure 2.1. *TCP segment format [3]*

Each field in the TCP segment indicates different functions. The details about each field are listed in Table 2.1:

Table 2.1. *The fields in a TCP segment [3]*

Field name	Number of bits	Meaning
Source port	16	Identify the process of sending packets
Destination port	16	Identify the process of receiving packets
Sequence number	32	If the SYN flag is not set, it indicates the position of the first byte of the data segment from the senders. If the SYN flag is set to 1, it indicates the initial sequence number in the senders' data streams.
Acknowledgment number	32	For the ACK flag is set to 1, it refers to the sequence number of next data byte to be sent by the sender. Otherwise, no meaning.
Header length	4	It states the length of the TCP header
Reserved	4	Not used, all of them are set to 0
Flags	8	Contain flag bits
Window	16	It specifies the size of the receive window which states the number of bytes to be accepted by the sender.
Checksum	16	To verify the integrity of the TCP segment received
Urgent pointer	16	It states how much urgent data are going to be sent
Options	Variable	This field is used to extend TCP functionality
Padding	Variable	To maintain a 32 bit boundary for the TCP header

In order to understand how TCP establish a connection between two hosts, it is good to know the fields in the flag field of TCP header. The fields of flag in the TCP header are shown as follows:

Table 2.2. *The fields of flag in the TCP header [3]*

Field name	Number of bits	Information
CWR (Congestion Window Reduced)	1	It is set by the sender to show that the sender has received a TCP segment with the ECE bit set
ECE (ECN-Echo)	1	It states that a host may use ECN when the connection is established
URG (Urgent)	1	When this field is set to 1, it means this segment contains urgent data.
ACK (Acknowledgment)	1	It indicates that ACK number field is valid
PSH (Push)	1	It means that the senders should send the data immediately
RST (Reset)	1	It is used to abort the connection
SYN (Synchronize)	1	To establish a connection
FIN (Finished) [4]	1	To terminate a TCP connection

2.3. Connection establishment

For transmitting the data reliably, it is important for TCP to establish a solid connection. So, how does TCP build a connection between two nodes? TCP uses 3-way handshake to establish a reliable connection. The process of a 3-way handshake is shown as follows:

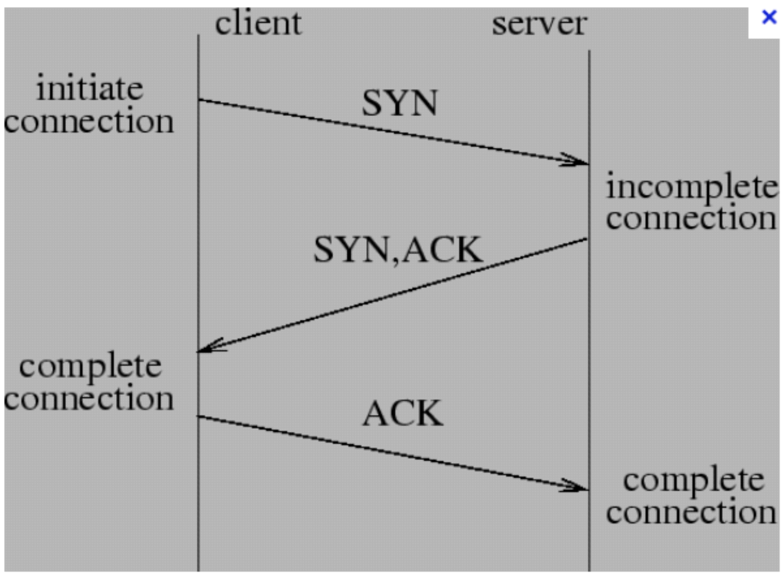


Figure 2.2. *The progress of the 3-way handshake [5]*

In the first handshake, the sender sends a special TCP segment to the receiver. This segment does not have any application data. It only has the TCP header with the SYN bit set to 1, so it is called a SYN segment. The SYN segment is used to synchronize the sender and receiver so that a stable connection can be built. The SYN segment is encapsulated in an IP packet and sent. When the SYN segment is accepted by the receiver, it will send a connection-granted segment back to the sender. So, this segment is also called SYN/ACK segment. This segment does still not contain any application data. In the connection-granted segment, the SYN bit is still set to 1, and the initial sequence number (ISN) is ISN (A) +1 in the acknowledgment number field. The receiver also sets its own initial sequence number ISN (B), and puts it into the sequence number field of the TCP header. After accepting the SYN/ACK segment, the sender will send the second segment to the receiver to acknowledge the SYN/ACK segment. This segment is to tell the receiver it is fine to establish a connection between them, so this segment may be called ACK segment. Then, the connection is established and the sender will send the data. At the same time, the SYN bit is set to 0 from 1 because the connection has been built. [6]

2.4. 802.11

802.11 is a set of telecommunication standards developed by IEEE to implement WLANs. Since World War II, wireless communication has been used widely more and more but there were no general standards to manage wireless communication protocols. So, in 1997, IEEE released the first 802.11 protocol for wireless local area networks. 802.11-1997 protocol is the original 802.11 protocol serving the WLANs; however, 802.11b is the first widely accepted protocol. Most of 802.11 protocols are developed from the original 802.11, and the most popular ones of them are 802.11b and 802.11g. Usually, the frequency bands where 802.11 protocols work are from 2.4 GHz to 5 GHz. [7]

Some major 802.11 protocols, not all of 802.11 protocols, are shown in the Table 2.3. More details about 802.11 protocols in Table 2.3 can be seen below the table:

Table 2.3. *Some major 802.11 protocols [7]*

Protocols	Release	Frequency (GHz)	Bandwidth (MHz)	Data rate per stream (Mbit/s)
802.11 (Original)	Jun 1997	2.4	20	1, 2
802.11a	Sep 1999	3.7 or 5	20	6, 9, 12, 18, 24, 36, 48, 54
802.11b	Sep 1999	2.4	20	1, 2, 5.5, 11
802.11g	Jun 2003	2.4	20	6, 9, 12, 18, 24, 36, 48, 54

802.11n	Oct 2009	2.4 or 5	20	7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2
			40	15, 30, 45, 60, 90, 120, 135, 150
802.11ac (Under development)	Nov 2011	5	80	433, 867
			160	867 Mbps, 1.73 Gbps, 3.47 Gbps, 6.93 Gbps

- 802.11

The first version of 802.11 is also called 802.11-1997 because it was released in 1997 and clarified in 1999. However, it is replaced by updated 802.11 protocols and is not used any more now. 802.11-1997 operated at 1 Mbps or 2 Mbps. The air interface modulation scheme that the 802.11-1997 takes is direct sequence spread spectrum (DSSS) and frequency hopping spread spectrum (FHSS). The maximum data rate of 802.11-1997 is 2 Mbit/s. [7]

- 802.11a

802.11a is an extension to the 802.11-1997 and it was released in 1999. Many versions of 802.11 work in 2.4 GHz band, which makes channels be crowded. As an improved version of original 802.11, the work frequency band for 802.11a is 5 GHz which improves the quality of wireless communication obviously. Nevertheless, high work frequency band also brings 802.11a a disadvantage: the effective cover range of 802.11a is less than other 802.11 protocols working on 2.4GHz frequency. The physical layer modulation method of 802.11a is orthogonal frequency division multiplexing (OFDM) rather than DSSS or FHSS. The maximum data rate of 802.11a is 54 Mbit/s. [7]

- 802.11b

802.11b was released in 1999 and it was improved directly from the original version of 802.11. It operates at the same frequency band with 802.11-1997 which is defined at 2.4 GHz. Since the decrease of the price and dramatic improved throughput, 802.11b was accepted widely and rapidly by the users. Although the 802.11b improves the quality of network transmission positively, it is still suffering from interferences because many other electronic devices or 802.11 protocols also work at 2.4 GHz, such as some microwave ovens, 802.11g\n, and so on. The physical layer modulation method of 802.11b is DSSS. The maximum data rate of 802.11b is 11 Mbit/s. [7]

- 802.11g

802.11g was released in 2003, and it is a version that combines some specifications from 802.11a and 802.11b. The 802.11g also works at 2.4GHz like the 802.11b while it also uses OFDM in its air interface like the 802.11a. The 802.11g is able to have higher data rate such as 54 Mbit/s. [7]

- 802.11n

802.11n was released in 2009. It develops the 802.11 protocols by adding three more multiple-input multiple-output (MIMO) antennas, which improves the throughputs compared to old protocols. The maximum bandwidth may be 40 MHz, not 20 MHz like before. Moreover, the maximum data transmission rate is improved to 150 Mbit/s. The air interface modulation scheme is OFDM just like 802.11a. [7]

- 802.11ac

802.11ac might be the latest developing version of 802.11 family. It aims to provide higher throughputs in the 5 GHz frequency band. As an improvement of 802.11n, the 802.11ac has 8 MIMO streams. The maximum bandwidth can reach to 160 MHz and the maximum data rate may be 6.93 Gbit/s. [7]

As shown in Table 2.3, the bandwidth of the latest 802.11 protocol is bigger and the data rate of 802.11 protocols is faster and faster. In 1999, a trade association called Wi-Fi Alliance was formed to operate wireless local area network brand named “Wi-Fi”. The 802.11 protocols support the Wi-Fi products. The wireless network products are allowed to use the brand “Wi-Fi” after they are certificated by the Wi-Fi alliance. [8] The WLAN provides wireless service to hosts so that the users can access the networks like Internet without any cable. By deploying WLANs and applying the 802.11 protocols, people can access the wireless networks seamlessly with any handy electric devices like smartphones, laptops, or tablets while they are travelling in museums, airports, or schools.

2.5. 802.11 operation modes

802.11 protocols have two kinds of operating mode: one is infrastructure mode, and the other one is Ad hoc mode. The configuration of the infrastructure mode is shown in Figure 2.3. In the infrastructure mode, the wireless devices access traditional wired networks like Internet through at least one wireless AP. The AP is a base station and is connected to the wired networks. The APs are responsible for managing data exchange between wireless and wired networks. [9]

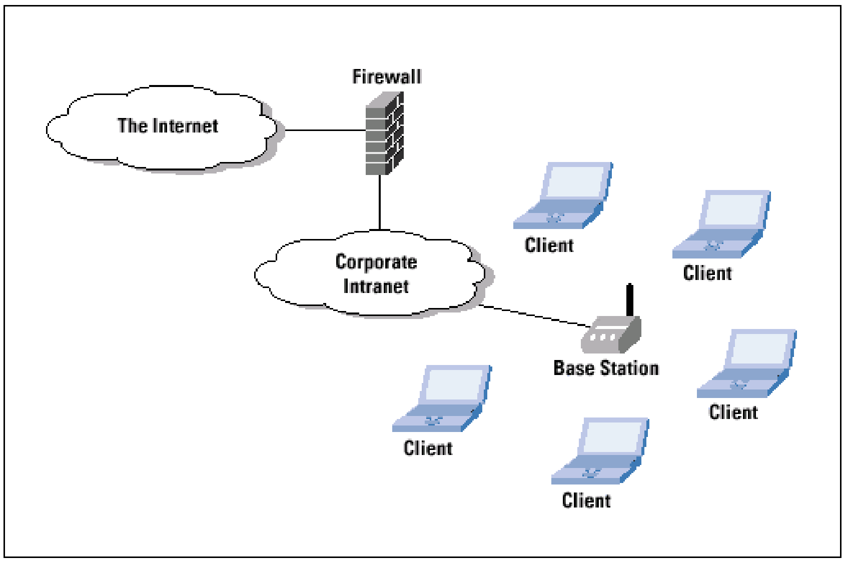


Figure 2.3. *802.11 configuration in infrastructure mode [10]*

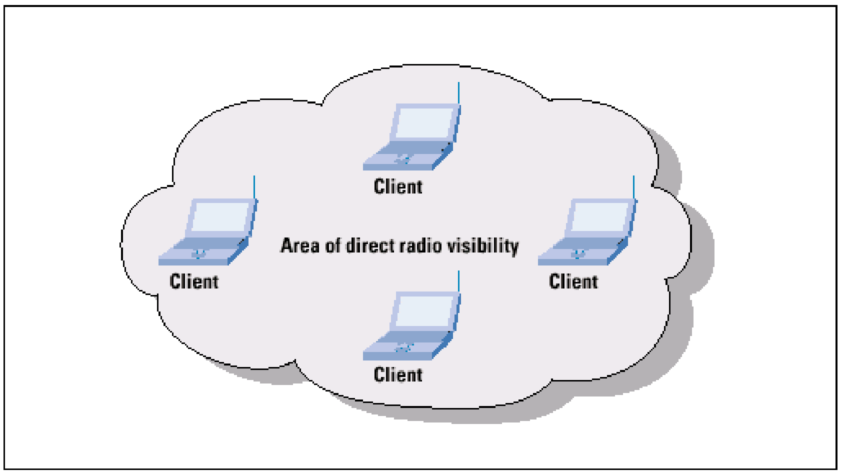


Figure 2.4. *802.11 configuration in Ad hoc mode [10]*

The configuration of the ad hoc mode is shown in Figure 2.4. In the ad hoc mode, the wireless clients communicate with each other directly without any APs or routers. The clients are equal peers which support 802.11 ad hoc mode. The clients in the ad hoc mode forward the data for other peers. The routines are operated by the hosts themselves in the ad hoc mode, and the routing is dynamic depending on the network connectivity. The ad hoc mode make the WLANs be easy to be deployed and flexible, which is very useful for some specific areas just like military. If a node is broken or destroyed, the network can fix itself by reorganizing the routing dynamically. [8]

In this paper, how an artificial bottleneck channel affecting the throughputs of different versions of TCP is researched. So, a set of wired-cum-wireless networks with base stations are designed. Due to every scenario having a base station, only the infrastructure mode is discussed here.

2.6. 802.11 MAC

802.11 protocols specify media access control (MAC) and physical (PHY) layers of the OSI model. 802.11 protocols use CSMA/CA as a method in MAC layer instead of

CSMA/CD to access the wireless channels [11]. Currently, most of 802.11 protocols use Distributed Coordination Function (DCF) scheme in CSMA/CA. In CSMA/CA with DCF, wireless nodes listen to the channel firstly for a Distributed Inter Frame Space (DIFS) interval when they want to send data packets. Then, if the channel is free, they will send the packets. If the packets are received, the receivers will send acknowledgment (ACK) frames after a Short Inter Frame Space (SIFS) interval. If the senders do not receive ACKs, the senders will think collisions have occurred during transmission. Once any sender confirms last packet is lost, it will send the same packet again when the channel is idle after another DIFS interval. [12] If the channel is still not free at the beginning, the node will wait for another random period of time until the channel is idle.

In the infrastructure mode, CSMA/CA with DCF enables all wireless nodes including the APs to share the wireless channels fairly. By CSMA/CA, the wireless nodes including base stations have the same priority to send their data packets in shared channels, which is quite important in this paper. [13]

3. PROBLEM STATEMENT

In former chapter, TCP and 802.11 are introduced separately. In some wireless-cum-wired network scenario like Figure 3.1, there are multiple wireless senders supporting 802.11 WLAN, multiple wired receivers, and only one base station. Here, uplink TCP flows (from wireless part to wired part) are sent from multiple wireless nodes and downlink ACKs (from wired part to wireless part) are sent by only one AP. All delivered data have to pass through the only base station. The scenarios like this bring one question: is it possible for a base station being a bottleneck for the uplink because of delay of ACKs in the downlink? If the answer is yes, there will be big difference about throughput between different versions of TCP with less and more ACKs.

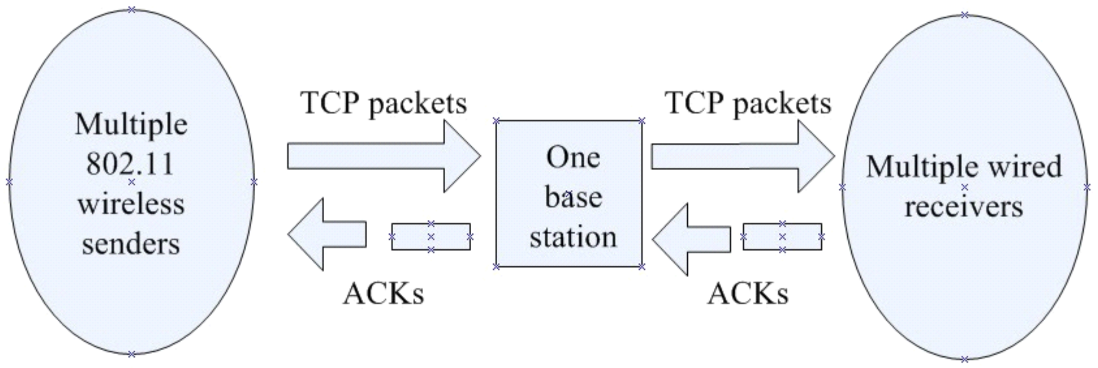


Figure 3.1.*The scenario where there are multiple wireless senders, multiple wired receivers and one base station*

Because of CSMA/CA with DCF, each wireless node is assigned theoretically the same priority to access the channel. Each of the wireless nodes has to wait for DIFS intervals and has the same right to access the wireless channel. All of the wireless nodes including the base stations have to compete for shared channel resources using CSMA/CA mechanism to get their fair share of bandwidth. [14] The wireless uplink traffic is formed by the senders, and the wireless downlink traffic is sent by the base station. If there are n wireless nodes and one AP accessing the wireless channel, the channel access probability of uplink is n/ (n + 1), and it is only 1/ (n + 1) for the downlink flow [14]. By this way, if there are 5 wireless nodes and one base station in some wireless-cum-wired network, the channel access probability of uplink is 5/6, and that of the downlink flows is only 1/6. If there are 40 wireless nodes and one base station in the wireless-cum-wired network, the channel access probability of uplink is 40/41, and it is 1/41 for the downlink. So, the channel access probability is bigger for the uplink flows when there are more wireless nodes in the network. However, the channel access probability of the downlink is less when the wireless nodes are more in the network. This may cause traffic congestion in the downlink, which affects the throughput of uplink.

The buffer size of base stations is limited. When uplink and downlink flows compete to access the base station, the packets will have to be buffered waiting for being delayed by the base station. Then, one of two situations happens possibly. The first one is that it is possible for ACKs to have more chances to access buffer space even though the channel access probability of the downlink is less because TCP only activate congestion control scheme against data packets. Further, it is possible that too many delayed ACKs take up too much buffer space, which causes uplink packets are dropped. The second situation is that it is easier for ACKs in the downlink to be delayed or dropped because small channel access possibility. If the senders do not receive enough ACKs, they will send the packets again, which makes the channel be more crowded. These two situations will cause the same result: the throughput of uplink decreases because the base station becomes a bottleneck for the uplink.

In fact, there are two possible results for throughput to use TCP with more ACKs and TCP with less ACKs: the throughput of uplink decreases obviously when more ACKs are transmitted in the downlink because of congestion in the channel, or the throughput of uplink is not changed obviously when more ACKs are transmitted in the downlink because of little competition time and packet size for the packets in the downlink. In order to get the practical result, some experiments have to be done through simulations. The purpose of the experiments is to find out whether there is any obvious difference for throughput of two different versions of TCP to use more or less ACKs. In the experiments, the performance of throughput of two versions of TCP protocol is compared: one can generate more ACKs and the other one can generate less ACKs. For the first version of TCP, normal version of TCP will be used. For second version of TCP, TCP with delayed ACKs will be used. In the normal version of TCP, the receivers will send ACKs immediately once they receive the packets sent by the senders. In TCP with delayed ACKs, the receivers will send ACKs after a period of time when they receive the data packets. If the parameter “interval_” is set to 200ms and two packets are received during 200ms, these two packets will be confirmed by one ACK. However, if only one packet is received and the timer expired, only this packet is confirmed. Moreover, ACKs in TCP with delayed ACKs can be cumulative if more than one data packet is received during the delay. [18] [19] Thus, the latter version of TCP generates less traffic (ACKs) in the downlink. More details about the experiments can be found in next chapter.

4. SIMULATIONS

In order to find out how big the difference is for different versions of TCP, their throughputs have to be compared. The throughputs can be calculated from the trace files generated from specific experiments which are a series of designed simulations. The experiment scenarios are simulated by using Network Simulator 2 (NS2). In this chapter, brief description of NS2 is presented firstly. Secondly, the simulation scenarios, process, and some parameters are introduced. Then, the trace files, throughput calculation, and AWK code are also stated.

4.1. Introduction to NS2

NS2 is widely utilized networking simulation software. Right now, the modules of NS2 almost cover all networking technologies. The users can use NS2 to simulate and research all kinds of networking protocols. Moreover, NS2 is also used in school education. The students learn networking technologies through utilization of NS2. Currently, many people have used or are using NS2.

NS2 is an object-oriented, discrete event driven network simulator. The Network Simulator was developed at UC Berkeley and it was written in C++ and OTcl [16]. NS2 has a virtual clock inside, and all simulations which are run by NS2 are driven by discrete events. It can be used to simulate wired, wireless, and wired-cum-wireless scenarios. NS2 is able to simulate a variety of networking protocols and functions, such as network transmission protocols, traffic generators, routing queue mechanism, routing algorithms, multicast, MAC protocols, and so on. [16] NS2 has a module called network animator and has a function being able to record and store link information into trace files [15]. So, the users can watch animation how the networks simulated works and analyze trace files after the simulations to figure out what happens in the networks.

NS2 is public network simulation software which is free of charge. It can be downloaded from official website. The release of NS2 used in this paper is version 2.35 which was released in November 4, 2011. Usually, NS2 runs in Linux environment. Fortunately, there is software called “Cygwin” which offers a Unix-like environment on Microsoft Windows platform [17]. The NS2 scripts in this paper are run in Unix-like environment supported by Cygwin.

4.1.1. Modules and simulation process of NS2

In order to simulate all kinds of networks, many function modules are encapsulated inside NS2. The typical modules include nodes, links, agents, data packets, and more. The nodes can represent source nodes, routers, or other nodes. The links connect nodes, deliver data packets, and manage transmission. The agents are attached on the nodes and are given port numbers. They generate or receive traffic. The data packets carry information transmitted on links. [16]

Generally, the progress of a NS2 simulation is as follows:

1. Find out what scenario is going to be simulated.
2. Start to write TCL scripts, setup parameters, and build topologies.
3. Run the scripts.
4. Get simulation results and analyze the trace files.
5. If new data are needed, repeat the process.

4.2. NS2 simulations of different versions of TCP in 802.11 WLANs

4.2.1. Scenario description

In order to verify theoretical assumption discussed in the chapter3, multiple simulations should be made through application of NS2. First of all, basic topology parameters should be determined. According to different types of TCP sinks, the simulations are classified into two groups: One is normal TCP, and the other one is TCP with delayed ACKs. For the first group, the NS2 parameter of receivers is Agent/TCPSink and it is Agent/TCPSink/DelAck for the second group. In the first group, the receivers will send ACKs immediately after they confirm the data packets sent by the senders. In the second group, the ACKs are delayed. The users use the parameter "interval_" to set delay time. In the simulations of this paper, two intervals are set for the TCP agents with DelAck: 100ms and 200ms. The delay time decides the amount of time generating an acknowledgment for data packets. [19] More topology parameters of two scenario groups may be seen in Table 4.1 and Table 4.2.

Table 4.1. *Basic topology parameters for the scenarios with Agent/TCPSink*

Agent	Number of sources	Buffer size (packets)
TCPSink	5	10
TCPSink	5	30
TCPSink	5	50
TCPSink	10	10
TCPSink	10	30
TCPSink	10	50
TCPSink	20	10
TCPSink	20	30
TCPSink	20	50
TCPSink	40	10

TCPSink	40	30
TCPSink	40	50

Table 4.2. *Basic topology parameters for the scenarios with Agent/TCPSink/DelAck*

Agent	Interval (milli-seconds)	Number of sources	Buffer size (packets)
TCPSink/DelAck	100	5	10
TCPSink/DelAck	100	5	30
TCPSink/DelAck	100	5	50
TCPSink/DelAck	100	10	10
TCPSink/DelAck	100	10	30
TCPSink/DelAck	100	10	50
TCPSink/DelAck	100	20	10
TCPSink/DelAck	100	20	30
TCPSink/DelAck	100	20	50
TCPSink/DelAck	100	40	10
TCPSink/DelAck	100	40	30
TCPSink/DelAck	100	40	50
TCPSink/DelAck	200	5	10
TCPSink/DelAck	200	5	30
TCPSink/DelAck	200	5	50
TCPSink/DelAck	200	10	10
TCPSink/DelAck	200	10	30
TCPSink/DelAck	200	10	50
TCPSink/DelAck	200	20	10
TCPSink/DelAck	200	20	30
TCPSink/DelAck	200	20	50
TCPSink/DelAck	200	40	10
TCPSink/DelAck	200	40	30
TCPSink/DelAck	200	40	50

As shown in Figure 4.1, the scenarios are wired-cum-wireless networks. On the left side of a scenario, there are 5, 10, 20, or 40 wireless source nodes. The sources may be any electronic devices which support 802.11 WLANs, such as laptops, smart phones, or tablet computers. The only base station is on their right side and all of the source nodes are associated with the base station. The base station is linked with a router by a wired link. The TCP sinks, receivers in the network, are connected with the router by using wired links. The sinks may be any electronic devices, such as computers or servers. Each source sends the data to its relevant sink. For example, the first source node sends the data to the first sink, and the last source node sends its data to the last sink.

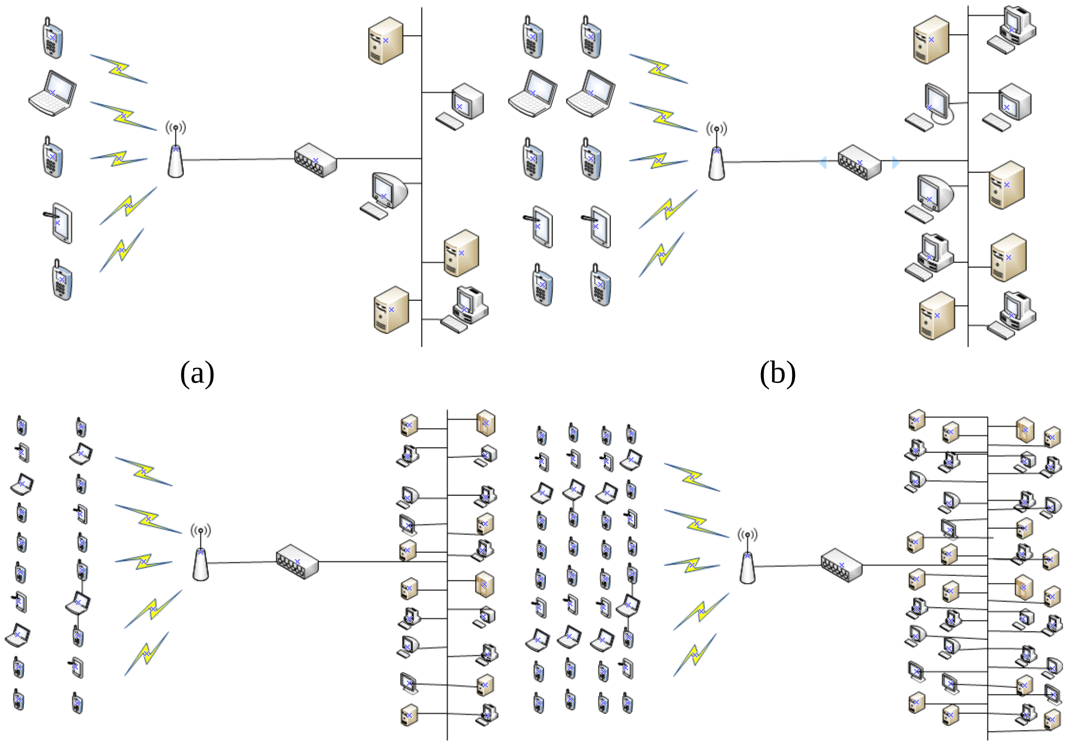

(a) (b) (c) (d)

Figure 4.1. *The simulation scenarios (a) the wired-cum-wireless network with 5 sources and 5 sinks (b) the wired-cum-wireless network with 10 sources and 10 sinks (c) the wired-cum-wireless network with 20 sources and 20 sinks (d) the wired-cum-wireless network with 40 sources and 40 sinks*

In the NS2 simulations, the source nodes will need some time to build the routines in the networks instead of sending the packets immediately. Otherwise, some packets probably will be abandoned accidently because the sources cannot find the routines to deliver the packets to the sinks. Further, this will definitely affect the throughputs calculated. After the routines are built, the source nodes send TCP packets carried by FTP traffic to their receivers. When the sinks confirm that the data packets arrive, they will send ACK packets back to the sources.

4.2.2. Simulation process and parameters

After the scenarios are built, the simulations can be started. In every simulation, firstly, the sources send TCP packets in FTP traffic through wireless environment when they are allowed to send the data. The base station receives and buffers TCP packets, and then delivers them to the wired router. Through wired links, the packets arrive at their own destinations. Once the receivers confirm the data packets have been received, they will send ACKs immediately or after a period of time depend on selected version of TCP. These ACKs go through the same way with the data packets in opposite direction. They also pass through and are buffered by the only base station. After a predetermined period of time, the sources stop sending the data finally and the simulation ends. At the moment, a trace file is generated and an animation file is made automatically. Take the scenario with 10 sources as an example, the result animation executed by NS2 is shown in Figure 4.2. After obtaining the trace files, the throughputs can be calculated by a written AWK file. The details about how to calculate throughput is introduced in the part 4.4, and the introduction to AWK can be found in part 4.5. Then, the throughputs are compared and analyzed. At last, the conclusion is obtained.

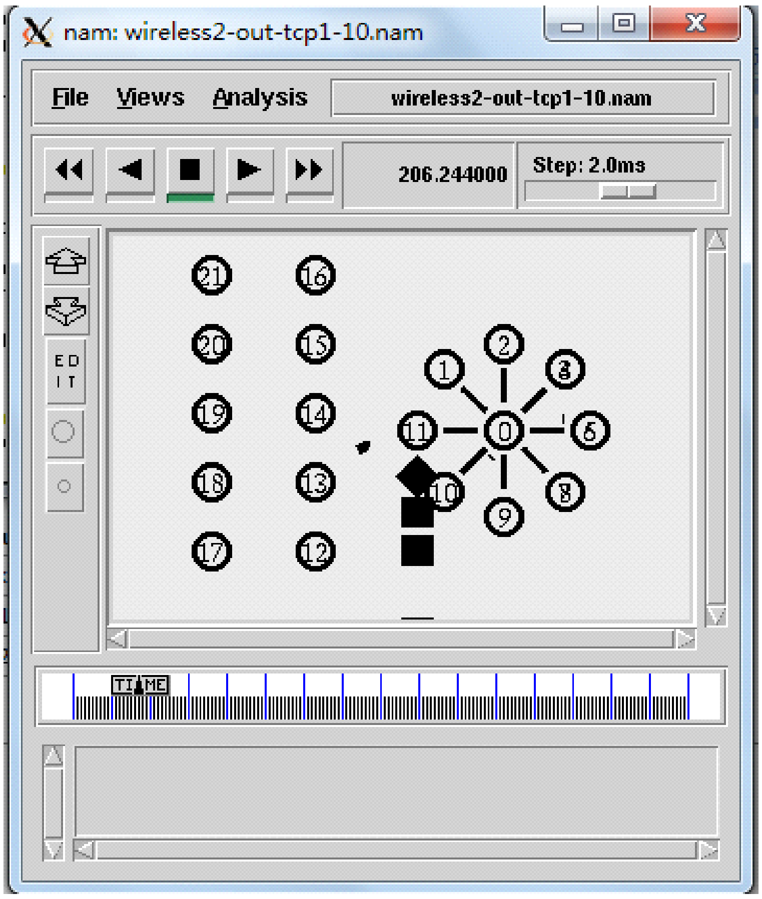


Figure 4.2. *The result animation executed of the scenario with 10 sources*

In the simulations, some important parameters which are listed in Table 4.3 have been determined. The specific scenario can be created by setting up the relevant parameters. The complete NS2 script can be found in Appendix 1, and the script can be run in NS2 directly.

Table 4.3. *Some important parameters in NS2 script [20]*

Code statement	Parameters	Meaning
set opt(netif)	Phy/WirelessPhy	Network interface type
set opt(mac)	Mac/802_11	MAC type
set opt(ifq)	Queue/DropTail/PriQueue	Interface queue type
set opt(ifqlen)	10/30/50	Buffer size of the base station
set opt(nn)	5/10/20/40	The number of mobile nodes
set opt(stop)	Variable	Time to stop the simulation
set num_wired_nodes	6/11/21/41	The number of wired nodes
set num_bs_nodes	1	The number of base station
AddrParams set domain_num_	2	Set up wireless and wired domains
lappend cluster_num	6/11/21/41 1	The number of clusters in each domain
set tcp	Agent/TCP	The type of senders
set ftp	Application/FTP	The type of traffic
set sink	Agent/TCPSink Agent/TCPSink/DelAck	The type of receivers
set interval_	100/200ms	Delayed time of ACKs

In addition, the users may use “lappend eilastlevel” to organize the nodes in wireless and wired domains. The code statement “set temp” is used to assign network addresses to wireless and wired nodes. [20] The start time and end time of FTP traffic are not fixed, because they need to be determined in line with practical situation. However, the simulation time for each scenario is fixed and it is always 1700 seconds. In the simulations, the packet size of TCP is 1000 bytes which is the default value in NS2. In terms of wired links, each link between the router and any receiver is a duplex link with the bandwidth 5 Megabits, a delay of 10 ms, and a DropTail queue. For the link between the base station and the router, it is a duplex link with the bandwidth 10 Megabits, a delay of 10 ms, and a DropTail queue. [20] More details about configuration of the simulations can be seen in Appendix 1.

4.3. Analysis of trace files

NS2 can trace and recode information concerning link activities in the simulations. In the TCL scripts, specific commands are needed to open a trace file. For example:

```
set net [open output.tr w]
$ns trace-all $net
```

Here, a variable related to a trace file named “net” is created. The trace file generated is named “output.tr”. The letter “w” means writing which refers to store the information into the trace file. The trace file will recode all information happening in the network which is needed by the users. [20] Depending on the network topology built is wired or wireless, the format of trace file is different. In our simulations, it is enough to just care about wired format. For wired networks, the general format of single trace is shown in Table 4.4:

Table 4.4. *NS2 wired trace format [21]*

Event	Time	Src node	Dst node	Packet type	Packet size	Flags	Flow id	Src addr	Dst addr	Seq num	Pkt id

Referring to the wired trace format, actual wired traces can be found in the trace files and they are like these:

```
r  2.068384 2 0 tcp 1060 ------- 0 1.0.1.0 0.1.0.0 1 4
+ 2.068384 0 1 tcp 1060 ------- 0 1.0.1.0 0.1.0.0 1 4
-  2.068384 0 1 tcp 1060 ------- 0 1.0.1.0 0.1.0.0 1 4
r  2.078689 2 0 tcp 1060 ------- 0 1.0.1.0 0.1.0.0 2 5
+ 2.078689 0 1 tcp 1060 ------- 0 1.0.1.0 0.1.0.0 2 5
-  2.078689 0 1 tcp 1060 ------- 0 1.0.1.0 0.1.0.0 2 5
```

Take the first trace as an example, it means that a TCP packet sent by node 2 is received by node 0 at 2.068384s. The letter “r” means that the packet is received by the receiver. If it is “d”, it means the packet is dropped. If it is “+” or “-”, it means the packet enters or leaves the queue. Besides, flow id of this packet is 0, sequence number is 1, and packet id is 4. The addresses also state the packet delivered goes through wireless and wired domains. [21]

4.4. Throughput calculation

In order to measure performance of different TCP versions in 802.11 WLANs, the throughput of each scenario should be calculated and compared. The formula to calculate throughput of some specific node can be represented as:

Throughput (Mbps) = the sum of packets received / effective time interval *8/1000000 [21]

In the simulations, the throughputs of some link between two specific nodes are cared about. In this paper, average throughput is needed. So, a series of throughputs for every scenario should be generated. The scripts to generate a series of throughputs and calculate the average throughput can be found in Appendix 2 and Appendix 3.

Because there is no packet being dropped on the wired link and only the data packets received are cared about, it is enough to just focus on the trace lines of wired network part in the trace files. To calculate throughput from a trace file generated by NS2 simulations, an AWK program is a good choice.

4.5. AWK description

In the simulations, the script to generate and calculate a series of throughputs is programmed by applying AWK language. AWK is a data process tool which processes text files to generate formatted reports. The name of AWK is formed by initial letters of family names of its creators --- Aho, Weinberger, and Kernighan. [22]

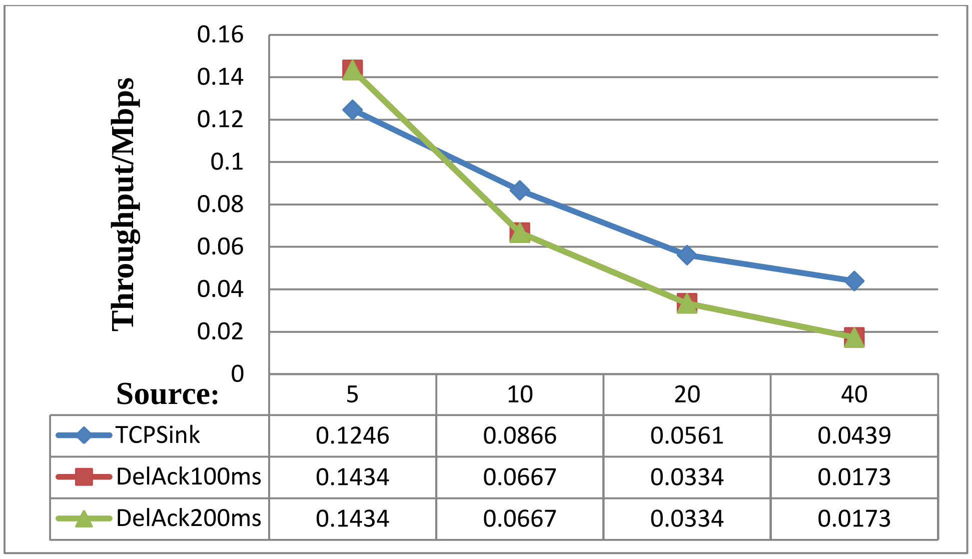
AWK can process trace files generated by NS2 simulations. It treats an NS2 trace file as a sequence of records. The advantage of AWK here is that it can recognize every trace line in a trace file and go through them one by one automatically. The AWK also can recognize the fields forming the trace line. After analyzing the data and calculating the throughputs, the AWK file will transfer the results to a specified text file. [22] More details about the AWK file used in the simulations can be found in Appendix 2.

5. RESULTS AND DISCUSSION

Through analyzing the trace files generated from the simulations, the throughputs can be calculated by using AWK code. By comparing the throughputs, it is convenient to find out how big the difference about the throughputs is for different versions of TCP which use more or less ACKs. Further, it is easy to find out whether the only base station may be a bottleneck for uplink because of delays of ACKs in downlink. The throughputs are classified into two groups: one is about the throughputs of different number of sources with the same buffer size, and the other one is about the throughputs of the same number of sources when the buffer size is different. The first group is composed of three throughput figures and three throughput histograms with confidence intervals, and the second group is composed of four throughput figures and four throughput histograms with confidence intervals.

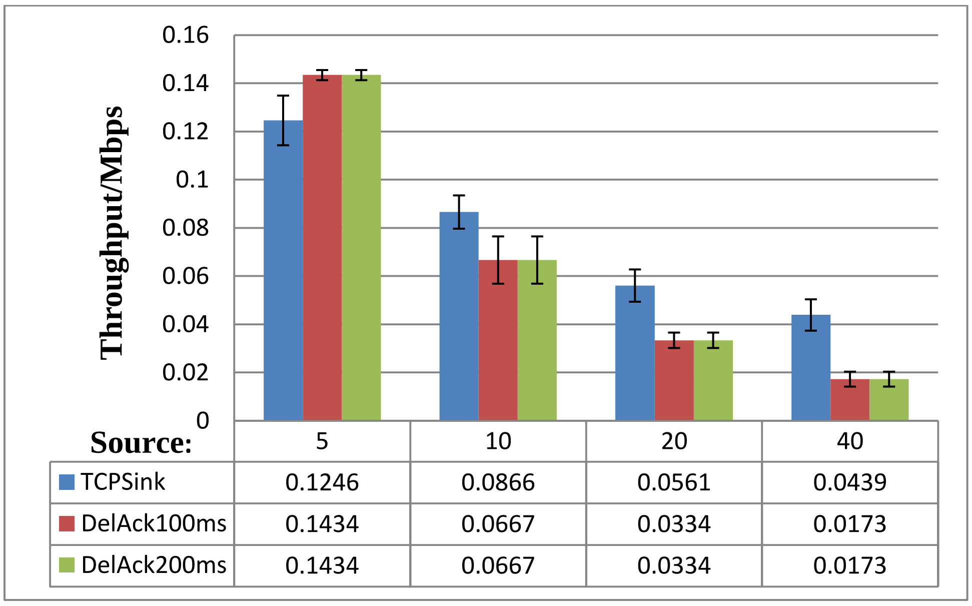
Confidence interval will be applied to descript the reliability of calculated throughputs of each figure in each group. Here, the confidence level is 95%, which means the possibility of real values is 95%. For example, if the confidence level is 95% and the confidence interval is between 0.9 Mbps and 1.1 Mbps for some scenario, this means it is 95% confident that the true value of the throughput is between 0.9 Mbps and 1.1 Mbps. [23] The code which is used to calculate the confidence intervals can be found in Appendix 3.

5.1. Throughputs for different number of sources with the same buffer size



Source:	5	10	20	40
TCPSink	0.1246	0.0866	0.0561	0.0439
DelAck100ms	0.1434	0.0667	0.0334	0.0173
DelAck200ms	0.1434	0.0667	0.0334	0.0173

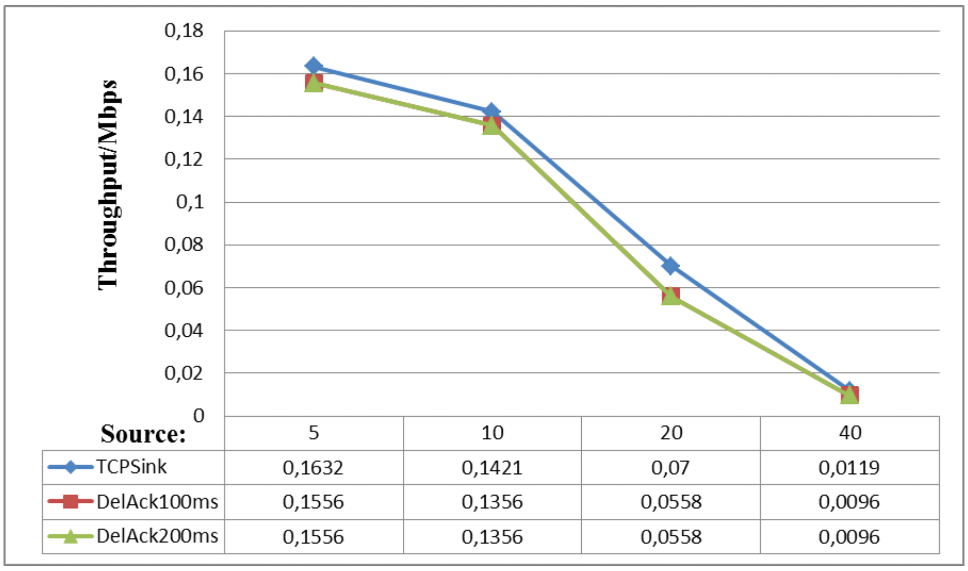
Figure 5.1. *Throughputs of different number of sources when the buffer size is 10 packets*



Source:	5	10	20	40
🟦 TCPSink	0.1246	0.0866	0.0561	0.0439
🟥 DelAck100ms	0.1434	0.0667	0.0334	0.0173
🟩 DelAck200ms	0.1434	0.0667	0.0334	0.0173

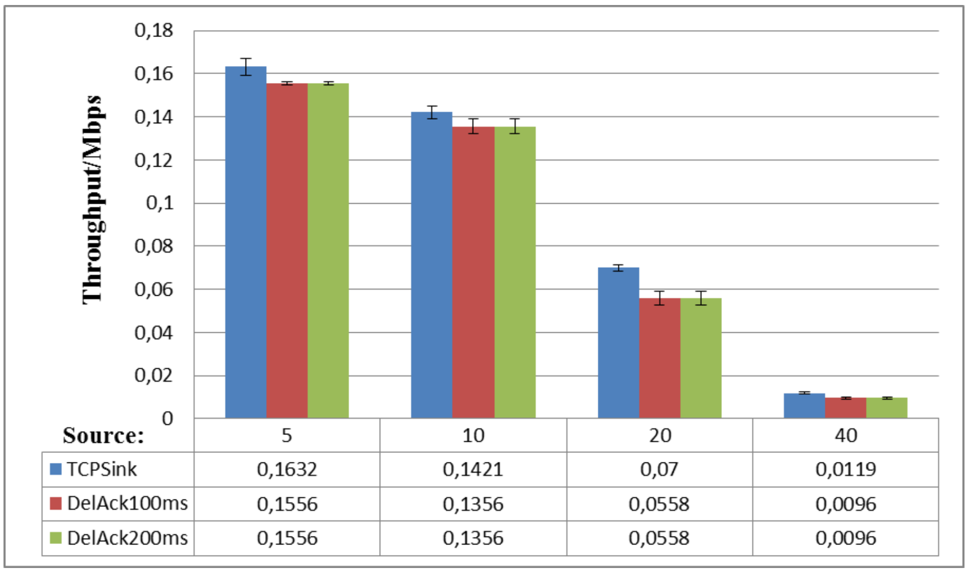
Figure 5.2. *Throughput histogram with confidence intervals when the buffer size is 10 packets*

The Figure 5.1 is a throughput figure for different number of sources with 10 packets in the buffer. The blue line is the throughput of TCP nodes without delayed ACKs, the red line is the throughput of TCP nodes with 100 ms delayed ACKs, and the green line is the throughput of TCP nodes with 200 ms delayed ACKs. From the figure, it is clear to see there is no big difference between the throughputs of different versions of TCP. The lines of TCP/DelAck 100ms and TCP/DelAck 200ms even completely overlap with each other, which indicate their throughputs are the same. For normal TCP and TCP with delayed ACKs, the difference about the throughputs can be seen but it is not too big so that the users have to decide to choose one of them. Talking about the distance between the lines in the figure, the biggest distance is 0.0266 Mbps when the number of sources is 10. Moreover, the trend of lines is similar: with increase of number of sources, the throughput of each TCP version decreases. The histogram with the confidence intervals shows the same change trend which is described in the Figure 5.1: the throughput of each version of TCP decreases when the number of sources increases. More importantly, the confidence intervals of the throughputs of different versions of TCP show clearly that it is not very big for the difference between the throughputs of different versions of TCP.



Source:	5	10	20	40
TCPSink	0,1632	0,1421	0,07	0,0119
DelAck100ms	0,1556	0,1356	0,0558	0,0096
DelAck200ms	0,1556	0,1356	0,0558	0,0096

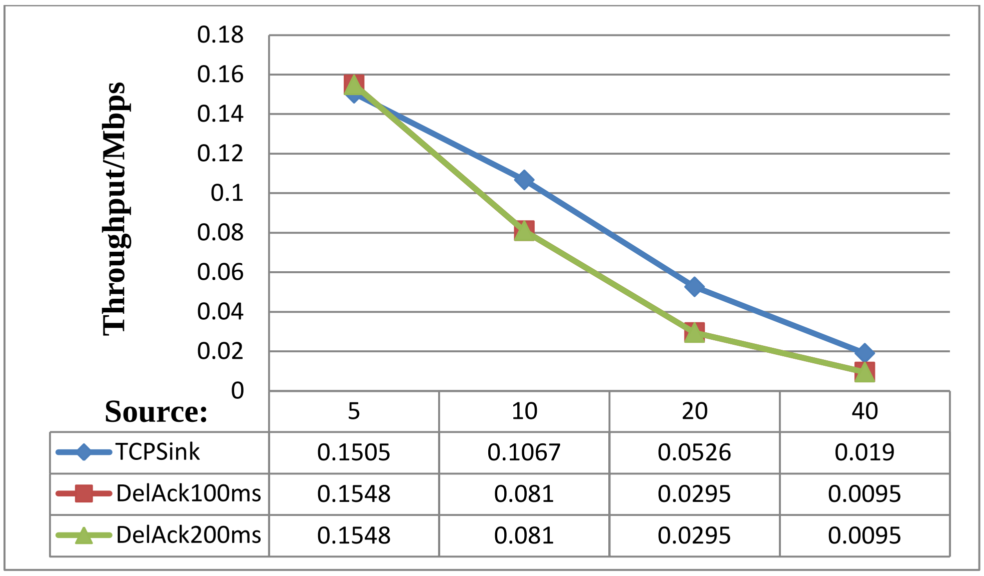
Figure 5.3. *Throughputs of different number of sources when the buffer size is 30 packets*



Source:	5	10	20	40
🟦 TCPSink	0,1632	0,1421	0,07	0,0119
🟥 DelAck100ms	0,1556	0,1356	0,0558	0,0096
🟩 DelAck200ms	0,1556	0,1356	0,0558	0,0096

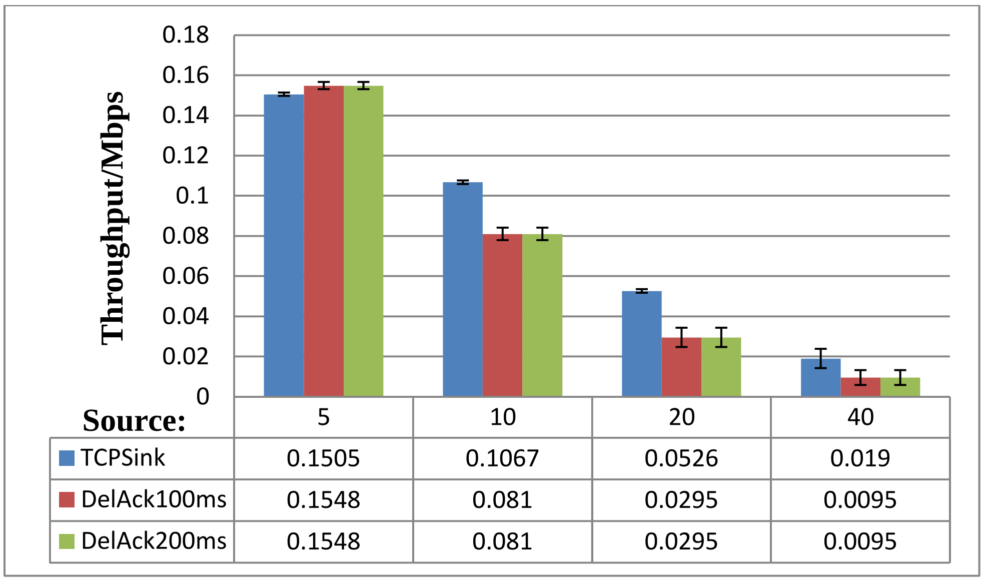
Figure 5.4. *Throughput histogram with confidence intervals when the buffer size is 30 packets*

The Figure 5.3 and Figure 5.4 are the throughput figure and the throughput histogram with confidence intervals for different number of sources with 30 packets in the buffer. These two figures indicate more clearly that there is no big difference about throughput between different versions of TCP. The biggest gap between lines is 0.0142 Mbps when the number of sources is 20. The confidence intervals of the throughputs of two versions of TCP also indicate the difference of throughputs of different versions of TCP is small. In fact, it is smaller than last figure. In addition, as shown in these two figures, the change trend of the throughputs still remains the same: the throughputs decrease when the number of sources increases. For TCP with 100ms and 200ms delayed ACKs, their throughputs are still the same.



Source:	5	10	20	40
TCPSink	0.1505	0.1067	0.0526	0.019
DelAck100ms	0.1548	0.081	0.0295	0.0095
DelAck200ms	0.1548	0.081	0.0295	0.0095

Figure 5.5. *Throughputs of different number of sources when the buffer size is 50 packets*

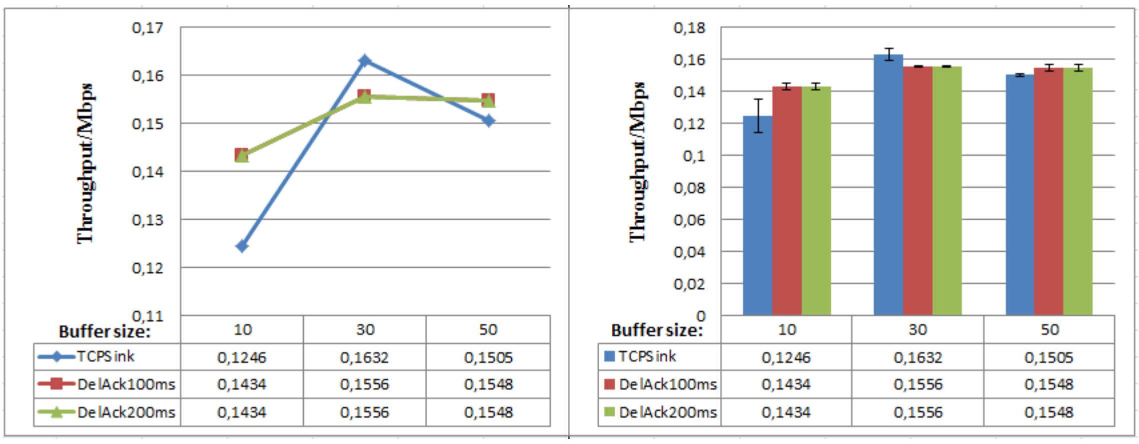


Source:	5	10	20	40
🟦 TCPSink	0.1505	0.1067	0.0526	0.019
🟥 DelAck100ms	0.1548	0.081	0.0295	0.0095
🟩 DelAck200ms	0.1548	0.081	0.0295	0.0095

Figure 5.6. *Throughput histogram with confidence intervals when the buffer size is 30 packets*

As shown in Figure 5.5 and Figure 5.6, just like the last two groups of figures, the difference between the throughputs of different versions of TCP is not big even the difference is noticeable. The biggest distance between the lines is 0.0257 Mbps when the number of sources is 10. From the confidence intervals, it also states that there is no big enough difference can be found for the throughputs of different versions of TCP. The throughputs of TCP with 100 and 200 ms delayed ACKs still overlap with each other. Moreover, the change trend of the throughputs does not change: the throughput decreases when the number of sources increases for each version of TCP.

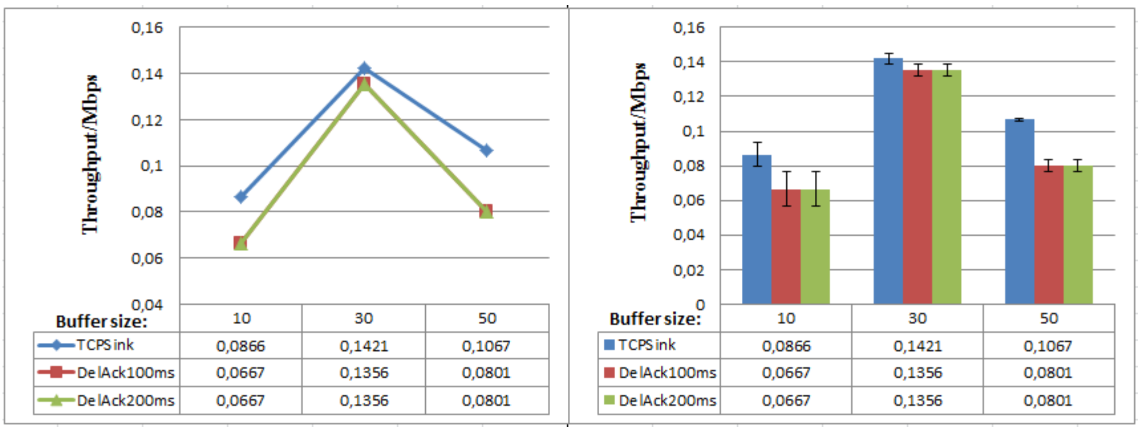
5.2. Throughputs for different buffer size when the number of sources is the same



Buffer size:	10	30	50
TCPSink	0,1246	0,1632	0,1505
DelAck100ms	0,1434	0,1556	0,1548
DelAck200ms	0,1434	0,1556	0,1548

Buffer size:	10	30	50
TCPSink	0,1246	0,1632	0,1505
DelAck100ms	0,1434	0,1556	0,1548
DelAck200ms	0,1434	0,1556	0,1548

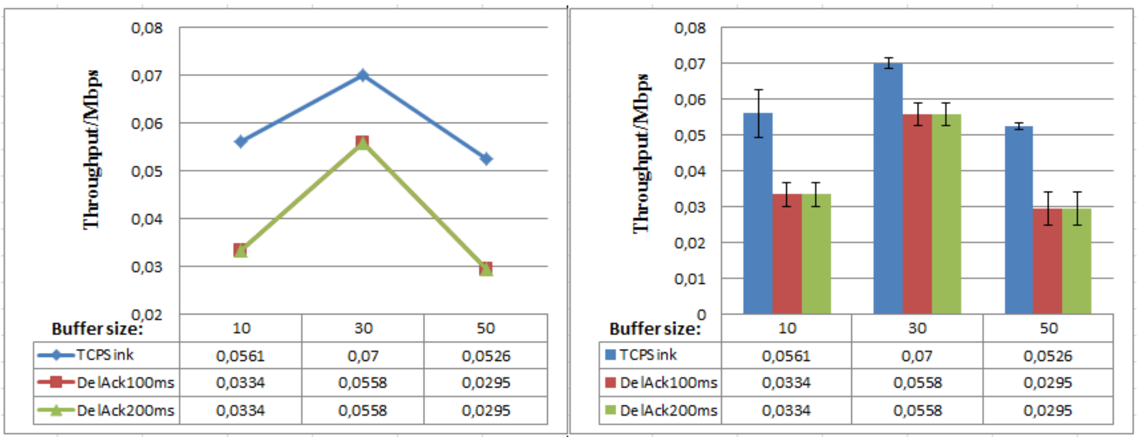
Figure 5.7. *Throughput figure and histogram for different buffer size when the number of sources is 5*



Buffer size:	10	30	50
TCPSink	0,0866	0,1421	0,1067
DelAck100ms	0,0667	0,1356	0,0801
DelAck200ms	0,0667	0,1356	0,0801

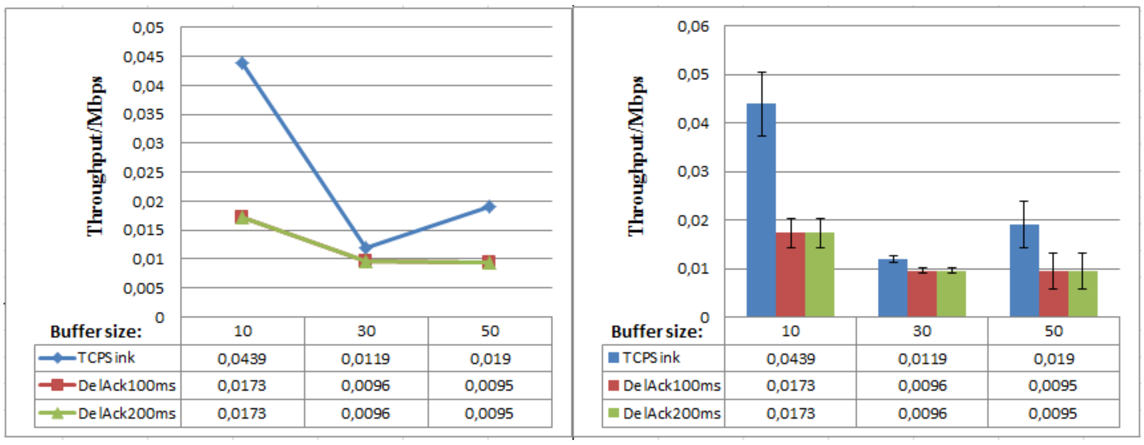
Buffer size:	10	30	50
TCPSink	0,0866	0,1421	0,1067
DelAck100ms	0,0667	0,1356	0,0801
DelAck200ms	0,0667	0,1356	0,0801

Figure 5.8. *Throughput figure and histogram for different buffer size when the number of sources is 10*



Buffer size:	10	30	50
TCPSink	0,0561	0,07	0,0526
DelAck100ms	0,0334	0,0558	0,0295
DelAck200ms	0,0334	0,0558	0,0295

Figure 5.9. *Throughput figure and histogram for different buffer size when the number of sources is 20*



Buffer size:	10	30	50
TCPSink	0,0439	0,0119	0,019
DelAck100ms	0,0173	0,0096	0,0095
DelAck200ms	0,0173	0,0096	0,0095

Figure 5.10. *Throughput figure and histogram for different buffer size when the number of sources is 40*

As shown in the figures above, firstly, it appears that the results are irregular for the same number of sources when the buffer size is different. The confidence intervals show the possible range where real values of the throughputs come from. When the buffer size is 10, 30, or 50 packets, it is extremely difficult for each figure to find any change pattern of the throughputs. It seems that the buffer size is irrelevant for the throughputs. So, it is possible that the throughputs can only be potentially affected by very small buffer size at the base station or exceptionally large number of competing sources in the network. However, such conditions rarely happen in daily life. Although it is impossible to find the change pattern for the throughputs, the figures and histograms still show there is no big difference for comparing the throughputs of different versions of TCP with different buffer sizes. For each scenario in any of the figures above, the difference of the throughputs is not big enough even the difference can be found. The biggest gap between the lines among all of the figures is 0.0266 Mbps when the number of sources is 40 and the buffer size of this scenario is 10 packets, which is still not big enough.

6. CONCLUSION

In this paper, the subject concerned is whether a base station in 802.11 WLAN can be a bottleneck for the uplink TCP flows because of ACKs in the downlink flows. 802.11 protocols use CSMA/CA in MAC layer to access wireless channels. CSMA/CA scheme enables all wireless sources and APs in 802.11 WLANs fairly share the wireless channels. In other words, each wireless node in 802.11 WLAN including any base station has the same priority to access the channel. Then, two situations may happen for an 802.11 WLAN which only has a base station: first, the only base station may become a bottleneck for the uplink flows because of delayed ACKs in the downlink flows. Second, the only base station may not become a bottleneck for the uplink flows because competition time for the packets in transmission is smaller than data sending time.

For studying the subject, some experiments have been done. The purpose of experiments is to find out whether there is any obvious difference for throughputs of two different versions of TCP to use more or less ACKs. So, a series of simulations are designed. In the simulations, two versions of TCP are used: one is normal TCP, and the other one is TCP with delayed ACKs which generate less ACKs than normal TCP during data transmission.

After analyzing the throughputs generated from the simulations, it can be found that the difference between the throughputs of TCP and TCP with delayed ACKs is not as drastic as some people might have expected although the difference is noticeable. As a matter of fact, the difference can be considered pretty small. In the scenarios of this paper, the users can ignore the difference and use any one of different versions of TCP. Whatever the number of sources is, the throughput of each version of TCP which generates more or less ACKs changes in the same way: the throughput decreases when the sources increase. For TCP with 100 ms and 200 ms delayed ACKs, their throughputs are the same, no any difference. In addition, from the simulations, it is known that the buffer size does not affect the throughputs obviously.

In the conclusion, it is obvious that the only base station in the 802.11 WLAN does not become a bottleneck for the uplink TCP flows because of ACKs in the downlink flows. Concerning the most possible reason which causes the results, it is because the competition time for the packets in transmission is smaller than data sending time. The reason is common fact. That causes the collisions are less than some users' assumption, which does not cause too many data packets being dropped. So, it is not always necessary for users to use TCP with less ACKs in wireless network communication. In daily life, normal TCP is good enough to be used in 802.11 WLANs.

REFERENCES

[1] WEBOPEDIA: TCP, http://www.webopedia.com/TERM/T/TCP.html

[2] WIKIPEDIA: Transmission Control Protocol, http://en.wikipedia.org/wiki/Transmission_Control_Protocol

[3] R. Dunaytesv, D. Moltchanov, TCP performance modeling in wired and wired/wireless networks: single source models. Dissertation. Tampere 2011. Tampe re University of Technology. Publication – LAP LAMBERT Academic Publishing. ISBN-13: 978-3-8465-0465-9, ISBN-10: 3846504653. 192 p.

[4] Firewall.cx: TCP Flag Options – Section 4, http://www.firewall.cx/general-topics-reviews/free-security-services/136-tcp-flag-options.html

[5] Usenix: TCP Denial of Service, http://static.usenix.org/events/bsdcon/full_papers/lemon/lemon_html/node2.html

[6] INETDAEMON: TCP 3-WAY HANDSHAKE (SYN, SYN-ACK, ACK) http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml

[7] WIKIPEDIA: IEEE 802.11, http://en.wikipedia.org/wiki/IEEE_802.11

[8] WIKIPEDIA: Wi-Fi, http://en.wikipedia.org/wiki/Wi-Fi

[9] Kioskea: WiFi modes of operation (802.11 or Wi-Fi), http://en.kioskea.net/contents/wifi/wifimodes.php3

[10] Internet Society: IEEE 802.11, http://www.isoc.org/pubs/int/cisco-1-1.html

[11] TechTarget: 802.11, http://searchmobilecomputing.techtarget.com/definition/80211

[12] M. Heusse, F. Rousseau, G. Berger-Sabbatel, A. Duda. Performance Anomaly of 802.11b [WWW]. [accessed on 3.4.2012]. Available at: http://www.cs.utexas.edu/~lili/classes/F11/reading2/5.pdf

[13] H. Chaouchi, A. Munaretto, G. Pujolle. Adaptive QoS Management for the IEEE 802.11 wireless networks [WWW]. [accessed on 3.4.2012]. Available at: http://www.infres.enst.fr/~demeure/WKSHP-ANWIRE/Chaouchi-slides.pdf

[14] J, Huang, J, Wang, J, Ye. A Buffer Management Algorithm for Improving Up/down TCP Fairness in IEEE 802.11 WLANs. International Journal of Com-munication Systems. 18 p.

[15] WPI: ns BY Example, http://nile.wpi.edu/NS/

[16] BAIDUBAIKE: NS2, http://baike.baidu.com/view/41867.htm

[17] WIKIPEDIA: Cygwin, http://en.wikipedia.org/wiki/Cygwin

[18] J. Chen, Y.Z. Lee, M. Gerla, M.Y. Sanadidi. TCP with Delayed Ack for Wireless Networks. Dissertation. Los Angelos. University of California. Publication - University of California. 10 p.

[19] ISI: Different agent objects, http://www.isi.edu/nsnam/ns/doc/node119.html

[20] ISI: ns tutorial, http://www.isi.edu/nsnam/ns/tutorial/index.html

[21] NS2 Learning Guide: Tool Introduction, http://140.116.164.80/~smallko/ns2/tool.htm

[22] AWK: A Tutorial and Introduction – by Bruce Barnett, http://www.grymoire.com/Unix/Awk.html

[23] Stat Trek: What is a Confidence Interval, http://stattrek.com/estimation/confidence-interval.aspx

APPENDIX 1: TCL SCRIPT FOR SIMULATIONS

```
#===================================================
# Define options
#===================================================
global opt
set opt(chan)         Channel/WirelessChannel          ;# channel type
set opt(prop)         Propagation/TwoRayGround         ;# radio-propagation model
set opt(netif)        Phy/WirelessPhy                  ;# network interface type
set opt(mac)          Mac/802_11                       ;# MAC type
set opt(ifq)          Queue/DropTail/PriQueue          ;# interface queue type
set opt(ll)           LL                               ;# link layer type
set opt(ant)          Antenna/OmniAntenna              ;# antenna model
set opt(ifqlen)       10
#set opt(ifqlen)      30
#set opt(ifqlen)      50                               ;# max packet in ifq

set opt(nn)           5                                ;# 5 mobile nodes
#set opt(nn)          10                               ;# 10 mobile nodes
#set opt(nn)          20                               ;# 20 mobile nodes
#set opt(nn)          40                               ;# 40 mobile nodes

set opt(adhocRouting)   DSDV                           ;# routing protocol

set opt(x)     200                                     ;# x coordinate of topology
set opt(y)     200                                     ;# y coordinate of topology

set opt(stop)   2000                                   ;# time to stop simulation

set num_wired_nodes      6                             ;# number of wired nodes
#set num_wired_nodes      11
#set num_wired_nodes      21
#set num_wired_nodes      41

set num_bs_nodes          1                            ;# number of base stations

# create simulator instance

set ns [new Simulator]

#First group
```

```
set tracefd                    [open wireless2-out-tcp1-5.tr w]
#set tracefd                   [open wireless2-out-tcp1-10.tr w]
#set tracefd                  [open wireless2-out-tcp1-20.tr w]
#set tracefd                  [open wireless2-out-tcp1-40.tr w]

set namtrace                   [open wireless2-out-tcp1-5.nam w]
#set namtrace                  [open wireless2-out-tcp1-10.nam w]
#set namtrace                  [open wireless2-out-tcp1-20.nam w]
#set namtrace                  [open wireless2-out-tcp1-40.nam w]
#Second group
#set tracefd                    [open wireless2-out-tcp2-5.tr w]
#set tracefd                    [open wireless2-out-tcp2-10.tr w]
#set tracefd                    [open wireless2-out-tcp2-20.tr w]
#set tracefd                    [open wireless2-out-tcp2-40.tr w]

#set namtrace                  [open wireless2-out-tcp2-5.nam w]
#set namtrace                  [open wireless2-out-tcp2-10.nam w]
#set namtrace                  [open wireless2-out-tcp2-20.nam w]
#set namtrace                  [open wireless2-out-tcp2-40.nam w]

$ns trace-all $tracefd

$ns namtrace-all-wireless $namtrace $opt(x) $opt(y)
set chan [new $opt(chan)]

proc finish {} {
   global ns monFile tracefd namtrace
   $ns flush-trace
   close $tracefd
   close $namtrace
   close $monFile
   exec nam wireless2-out-tcp1-5.nam &
   #exec nam wireless2-out-tcp1-10.nam &
   #exec nam wireless2-out-tcp1-20.nam &
   #exec nam wireless2-out-tcp1-40.nam &
   #exec nam wireless2-out-tcp2-5.nam &
   #exec nam wireless2-out-tcp2-10.nam &
   #exec nam wireless2-out-tcp2-20.nam &
   #exec nam wireless2-out-tcp2-40.nam &
   exit 0
   }
```

```
# set up for hierarchical routing

$ns node-config -addressType hierarchical
AddrParams set domain_num_ 2            ;# number of domains
lappend cluster_num 6 1                 ;# number of clusters in each domain
#lappend cluster_num 11 1
#lappend cluster_num 21 1
#lappend cluster_num 41 1

AddrParams set cluster_num_ $cluster_num

lappend eilastlevel 1 1 1 1 1 1 6       ;# number of nodes in each cluster
#lappend eilastlevel 1 1 1 1 1 1 1 1 1 1 1 11
#lappend eilastlevel 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 21
#lappend eilastlevel 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 41

AddrParams set nodes_num_ $eilastlevel              ;# for each domain

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)

# create God
create-god [expr $opt(nn) + $num_bs_nodes]

#create wired nodes
#hierarchical addresses for wired domain

set temp {0.0.0 0.1.0 0.2.0 0.3.0 0.4.0 0.5.0}
#set temp {0.0.0 0.1.0 0.2.0 0.3.0 0.4.0 0.5.0 0.6.0 0.7.0 0.8.0 0.9.0 0.10.0}
#set temp {0.0.0 0.1.0 0.2.0 0.3.0 0.4.0 0.5.0 0.6.0 0.7.0 0.8.0 0.9.0 0.10.0 0.11.0 0.12.0
0.13.0 0.14.0 0.15.0 0.16.0 0.17.0 0.18.0 0.19.0 0.20.0}
#set temp {0.0.0 0.1.0 0.2.0 0.3.0 0.4.0 0.5.0 0.6.0 0.7.0 0.8.0 0.9.0 0.10.0 0.11.0 0.12.0
0.13.0 0.14.0 0.15.0 0.16.0 0.17.0 0.18.0 0.19.0 0.20.0 0.21.0 0.22.0 0.23.0 0.24.0
0.25.0 0.26.0 0.27.0 0.28.0 0.29.0 0.30.0 0.31.0 0.32.0 0.33.0 0.34.0 0.35.0 0.36.0
0.37.0 0.38.0 0.39.0 0.40.0}

for {set i 0} {$i < $num_wired_nodes} {incr i} {
   set W($i) [$ns node [lindex $temp $i]]
}

# conFigure for base-station node

$ns node-config -adhocRouting $opt(adhocRouting) \
                -llType $opt(ll) \
                -macType $opt(mac) \
                -ifqType $opt(ifq) \
                -ifqLen $opt(ifqlen) \
                -antType $opt(ant) \
                -propType $opt(prop) \
                -phyType $opt(netif) \
                -channel $chan \
                -topoInstance $topo \
                -wiredRouting ON \
                -agentTrace ON \
                -routerTrace OFF \
                -macTrace ON \
                -movementTrace OFF

#create base-station node and wired nodes

set temp {1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5}
#set temp {1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8 1.0.9 1.0.10}
#set temp {1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8 1.0.9 1.0.10 1.0.11 1.0.12
1.0.13 1.0.14 1.0.15 1.0.16 1.0.17 1.0.18 1.0.19 1.0.20}
#set temp {1.0.0 1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8 1.0.9 1.0.10 1.0.11 1.0.12
1.0.13 1.0.14 1.0.15 1.0.16 1.0.17 1.0.18 1.0.19 1.0.20 1.0.21 1.0.22 1.0.23 1.0.24
1.0.25 1.0.26 1.0.27 1.0.28 1.0.29 1.0.30 1.0.31 1.0.32 1.0.33 1.0.34 1.0.35 1.0.36
1.0.37 1.0.38 1.0.39 1.0.40}

set BS(0) [$ns node [lindex $temp 0]]
$BS(0) random-motion 0                          ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(0) set X_ 50.0
$BS(0) set Y_ 50.0
$BS(0) set Z_ 0.0

# create mobilenodes in the same domain as BS(0)

#conFigure for mobilenodes
```

```
$ns node-config -wiredRouting OFF

for {set j 0} {$j < $opt(nn)} {incr j} {
    set node_($j) [ $ns node [lindex $temp \
                        [expr $j+1]] ]
    $node_($j) base-station [AddrParams addr2id \
                        [$BS(0) node-addr]]
}

# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes

for {set i 0} {$i < 5} {incr i} {
    $node_($i) set X_ 20
    $node_($i) set Y_ [expr 15+$i*20]
    $node_($i) set Z_ 0.0
}
#for {set i 5} {$i < 10} {incr i} {
#   $node_($i) set X_ -10
#    $node_($i) set Y_ [expr ($i*20)-85]
#    $node_($i) set Z_ 0.0
#}
#for {set i 10} {$i < 15} {incr i} {
#    $node_($i) set X_ -40
#    $node_($i) set Y_ [expr ($i*20)-185]
#    $node_($i) set Z_ 0.0
#}
#for {set i 15} {$i < 20} {incr i} {
#    $node_($i) set X_ -70
#    $node_($i) set Y_ [expr ($i*20)-285]
#    $node_($i) set Z_ 0.0
#}
#for {set i 20} {$i < 30} {incr i} {
#    $node_($i) set X_ -40
#    $node_($i) set Y_ [expr ($i*20)-385]
#    $node_($i) set Z_ 0.0
#}
#for {set i 30} {$i < 40} {incr i} {
#    $node_($i) set X_ -70
#    $node_($i) set Y_ [expr ($i*20)-585]
#    $node_($i) set Z_ 0.0
#}
```

```
#Create agents and attach them to senders and receivers

for {set i 0} {$i < $opt(nn)} {incr i} {
   set tcp($i) [new Agent/TCP]
   $ns attach-agent $node_($i) $tcp($i)
   $tcp($i) set packetSize_ 1000
}

#Set up ftp traffic

for {set i 0} {$i < $opt(nn)} {incr i} {
   set ftp($i) [new Application/FTP]
   $ftp($i) attach-agent $tcp($i)
   $ftp($i) set type_ FTP
}

for {set i 0} {$i < $num_wired_nodes-1} {incr i} {
    set sink($i) [new Agent/TCPSink]

    #For TCP with delayed ACks
    #set interval_ 100ms
    #set interval_ 200ms
    #set sink($i) [new Agent/TCPSink/DelAck]

   $ns attach-agent $W([expr $i+1]) $sink($i)
}

#Connect agents between source and destination

for {set i 0} {$i < $opt(nn)} {incr i} {
   $ns connect $tcp($i) $sink($i)
}

#create links between wired and BS nodes

for {set i 1} {$i < $num_wired_nodes} {incr i} {
   $ns duplex-link $W($i) $W(0) 5Mb 10ms DropTail
}

$ns duplex-link $W(0) $BS(0) 10Mb 10ms DropTail
$ns duplex-link-op $W(0) $BS(0) orient left
$ns duplex-link-op $W(0) $W(1) orient left-up
```

```
$ns duplex-link-op $W(0) $W(2) orient up
$ns duplex-link-op $W(0) $W(3) orient right-up
$ns duplex-link-op $W(0) $W(4) orient right
$ns duplex-link-op $W(0) $W(5) orient right-down
#$ns duplex-link-op $W(0) $W(6) orient right
#$ns duplex-link-op $W(0) $W(8) orient right-down
#$ns duplex-link-op $W(0) $W(7) orient right-down
#$ns duplex-link-op $W(0) $W(9) orient down
#$ns duplex-link-op $W(0) $W(10) orient left-down
#$ns duplex-link-op $W(0) $W(11) orient left-up
#$ns duplex-link-op $W(0) $W(12) orient up
#$ns duplex-link-op $W(0) $W(13) orient right-up
#$ns duplex-link-op $W(0) $W(14) orient right-up
#$ns duplex-link-op $W(0) $W(15) orient right
#$ns duplex-link-op $W(0) $W(16) orient right
#$ns duplex-link-op $W(0) $W(18) orient right-down
#$ns duplex-link-op $W(0) $W(17) orient right-down
#$ns duplex-link-op $W(0) $W(19) orient down
#$ns duplex-link-op $W(0) $W(20) orient left-down
#$ns duplex-link-op $W(0) $W(21) orient left-up
#$ns duplex-link-op $W(0) $W(22) orient up
#$ns duplex-link-op $W(0) $W(23) orient right-up
#$ns duplex-link-op $W(0) $W(24) orient right-up
#$ns duplex-link-op $W(0) $W(25) orient right
#$ns duplex-link-op $W(0) $W(26) orient right
#$ns duplex-link-op $W(0) $W(27) orient right-down
#$ns duplex-link-op $W(0) $W(28) orient right-down
#$ns duplex-link-op $W(0) $W(29) orient down
#$ns duplex-link-op $W(0) $W(30) orient left-down
#$ns duplex-link-op $W(0) $W(31) orient left-up
#$ns duplex-link-op $W(0) $W(32) orient up
#$ns duplex-link-op $W(0) $W(33) orient right-up
#$ns duplex-link-op $W(0) $W(34) orient right-up
#$ns duplex-link-op $W(0) $W(35) orient right
#$ns duplex-link-op $W(0) $W(36) orient right
#$ns duplex-link-op $W(0) $W(37) orient right-down
#$ns duplex-link-op $W(0) $W(38) orient right-down
#$ns duplex-link-op $W(0) $W(39) orient down
#$ns duplex-link-op $W(0) $W(40) orient left-down

#Monitor the queue for link ($W(0) $BS(0) ). (for NAM)
```

```
$ns duplex-link-op $W(0) $BS(0) queuePos 1.5

# Define initial node size in nam

for {set i 0} {$i < $opt(nn)} {incr i} {
   $ns initial_node_pos $node_($i) 10
}

# Tracing the queue

set monFile [open mon.tr w]
$ns trace-queue $W(0) $BS(0) $monFile

# Starting and stopping sources

for {set i 0} {$i < $opt(nn)} {incr i} {
   $ns at 300.0 "$ftp($i) start"
}

for {set i 0} {$i < $opt(nn)} {incr i} {
   $ns at $opt(stop).0 "$ftp($i) stop"
}

# Tell all nodes when the simulation ends

for {set i 0} {$i < $opt(nn) } {incr i} {
   $ns at $opt(stop).0 "$node_($i) reset";
}

$ns at $opt(stop).0 "$BS(0) reset";

$ns at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns halt"
$ns at $opt(stop).0001 "finish"

#Start the simulation
puts "Starting Simulation..."
$ns run
```

APPENDIX 2: AWK FILE TO CALCULATE THE THROUGHPUTS

```
BEGIN {
            init=0;
            i=0;
            pkt_byte_sum[i] = 0;
            loop=0;
            interval=0;
}

{
          #action = $1;
          time = $2;
          #node_1 = $3;
          #node_2 = $4;
          #src = $5;
          pktsize = $6;
          #flow_id = $8;
          #node_1_address = $9;
          #node_2_address = $10;
          #seq_no = $11;
          #packet_id = $12;

        if ($1=="r" && $4 == 2 && $5== "tcp" && $6==1060 && $9 == "1.0.2.0"
&& $10 == "0.2.0.0") {

             pkt_byte_sum[i+1]=pkt_byte_sum[i]+ pktsize;

              if(init==0) {
                  start_time = time;
                  init = 1;
              }
               end_time[i] = time;
               i = i+1;
        }
}

END {

for (loop=1;loop < 35; loop++)  {
   for (j=1; j<i; j++){
       interval = 50 * loop;
       time1 = end_time[j]-300;

   if (time1 >= interval){
       th[loop] = (pkt_byte_sum[j-1] / interval)*8/1000000;
```

```
        #For final throughput
        #th = (pkt_byte_sum[i-1] / 1700)*8/1000000;

printf("%.4f\n", th[loop]);
break;   }

}
}
}
```

APPENDIX 3: MATLAB CODE TO CALCULATE CONFIDENCE INTERVALS

```
clear all
close all

[th]=textread('CI.txt','%f');
n=34;
average=mean(th)
sig=std(th);
high=average+1.96*(sig/sqrt(n))
low=average-1.96*(sig/sqrt(n))
CI=high-average
```