2014

Experimental evaluation of QCSMA and CSMA for ad-hoc wireless sensor networks on the Telosb platform

Bhavani Satyanarayana Rao
Iowa State University

Experimental evaluation of QCSMA and CSMA for ad-hoc wireless sensor networks on the Telosb platform

by

Bhavani Satyanarayana Rao

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:

Lei Ying, Major Professor

Daji Qiao

Wensheng Zhang

Iowa State University

Ames, Iowa

2014

DEDICATION

To my beloved family and friends

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor Dr. Lei Ying for his constant guidance and motivation throughout my research work. A major part of the research work for this thesis was done off-campus and would not have been possible without Dr. Ying's support and availability to guide me through it.

I would like to thank Dr. Daji Qiao for providing me with his invaluable perspective on conducting the experiments and analyzing the results in wireless sensor networks. It was Dr. Qiao's coursework on wireless sensor networks that provided me with the skills required to implement the link scheduling algorithms on the Telosb platform.

I am also thankful to Dr. Wensheng Zhang for spending his precious time in reviewing and providing me with valuable feedback on the writing of this thesis.

I am grateful to Dr. Shihuan Liu, former student of Dr. Ying, for helping me form ideas on how best to implement the CSMA and QCSMA algorithms on the Telosb platform using the TinyOS tool chain.

Finally, I would like to give my heartfelt thanks to my family and friends for being my constant source of motivation in completing this thesis, and for blessing me with their unyielding love and support.

ABSTRACT

Link Scheduling is to allocate limited resources in order to achieve high performance and acceptable Quality of Service (Jian Bo Srikant). A good scheduling algorithm is one which is easy to implement and acheives good performance in terms of throughput and delay. Maximal Weight Scheduling (MWS), a queue length based scheduling introduced in (Tassiulas Ephremides) is shown to be throughput optimal but is complex to be implemented in non-centralized wireless sensor system. Whereas, a Carrier Sense Multiple Access (CSMA) based random access scheduling algorithm such as that in (Boorstyn Kershenbaum Maglaris Sahin) can be easily implemented in a distributed system, shown to be throughput-optimal in idealised conditions but suffers from bad delay performance in simulations. (Jian Bo Srikant) proposed Queue length based CSMA (Q-CSMA) scheduling algorithm, which combines the two classes of scheduling algorithms and evaluates the performance of QCSMA with several queue length based scheduling algorithms in simulations.

The objective of this thesis is to implement the discrete time version of CSMA and Q-CSMA algorithms in a wireless sensor platform, to build a wireless sensor network on this platform and to evaluate the performance of the two algorithms practically. Telosb, an open source sensor node platform is used for this work.(Crossbow). In this thesis, an issue of data collision between the contending nodes in a real time network is observed and a solution to counteract this problem is proposed and its performance is analysed.

CHAPTER 1. Overview

This chapter sheds light on the importance of link scheduling in ad-hoc wireless sensor networks and gives an overview on distributed link scheduling algorithms or MAC protocols. It then outlines the motivation for this work and contributions and organization of this thesis.

1.1 Need for Link scheduling

Wireless sensor networks (WSNs) are an emerging technology with multitude of use in military, industrial and commercial applications. WSNs consist of autonomous sensors performing sensing (gathering data), computation (processing data) and communication. The sensor nodes communicate data with each other and to other communication network(s) or to a central base station. The communications in WSNs is mostly ad hoc without a central entity scheduling transmissions. Also, in WSNs, the electromagnetic signals propagated on one link may interfere with the other links unlike in the case of wired medium. In WSNs, link scheduling has distributed implementation and arbitrates the contentions in multiple links for a medium channel access. The design of these scheduling algorithms is based on the assumption that the flow of data into the network is constrained within the capacity region by the congestion control algorithms but in practice the scheduling algorithms affect the behavior of the congestion controlled algorithms (Srikant Lei).

1.2 Link scheduling algorithms

In wireless networks the sharing of wireless spectrum resource can be done in time (TDMA), frequency (FDMA), code (CDMA) or at random. ALOHA, ALOHA-S, CSMA, CSMA/CD, CSMA/CA are examples of channel partitioning MAC protocols based on randomized access. Al-

though CSMA based MAC protocols can be easily implemented in wireless sensor networks in a distributed fashion and are shown to be throughput optimal (Boorstyn Kershenbaum Maglaris Sahin) (Wang Kar), the delay experienced by packets in such networks can be quite large compared to other networks using Max weight scheduling based MAC protocols as discussed in (Jian Bo Srikant). Another class of scheduling algorithms based on simple heuristics such as distributed approximation of Greedy Maximal Scheduling (GMS) or Longest Queue First (LQF) can achieve better delay but for networks that do not satisfy a condition called local pooling, it can achieve only a fraction of the throughput.(Jian Bo Srikant). QCSMA combines the carrier sensing capability of CSMA and has queue length based link activation probabilities in order to achieve better delay and high throughput. (Jian Bo Srikant).

1.3 Motivation

The performance gain in terms of high throughput and low delay of QCSMA compared to CSMA type random access algorithm has been evaluated only in simulations. (Jian Bo Srikant). There is a need to evaluate and compare the performance of QCSMA with respect to CSMA closer to real world where the network parameters such as link loss and noise ratio are non-idealised.

1.4 Contribution

The contribution of this thesis is three-fold.

1. Implementation of QCSMA and CSMA scheduling algorithm for TinyOS operating system running on Telosb platform.

2. Building a self synchronized wireless network of Telosb sensor nodes.

3. Conducting experiments to evaluate and compare the performance of the two scheduling algorithms.

1.5 Organization

The remainder of this thesis is organized as follows. Chapter 2 provides a background on the network model and basic scheduling algorithms. This also provides an overview of sensor node platform used in this work. Chapter 3 details the QCSMA and CSMA algorithm. Chapter 4 contains the implementation details of QCSMA and discrete time version of CSMA/CA on Telosb. Chapter 5 provides the details on building a self synchronized network used in the experiments for this work. Chapter 6 presents the experiments conducted to evaluate and compare the performance of QCSMA and CSMA/CA and compares the two scheduling algorithms based on the result .It also contains the conclusion for this work and future work.

CHAPTER 2. Background

This chapter explains concepts, definitions required to understand link scheduling in wireless networks. This is followed by details on the sensor mote platform used for this work.

2.1 Network Model

The network model used for the design of the scheduling algorithm is from (Jian Bo Srikant). The model is a graph G(V,E) where V is the set of wireless transmitter/receiver nodes and E is the set of edges/links connected the nodes who can hear each other transmit. The assumption here is that if node a can hear node b (denoted by edge (a,b)) transmit then node b can also hear node a ie., (b,a)=(a,b). Links are of unit capacity and the time is discrete/slotted. The packets are generated at every node with a probability of λ known as the arrival rate.

A conflict set C(i) is defined as the set of links in G that cannot transmit while i is active. Conflict set C(i) satisfies two important constraints

1. node-exclusive constraint - no other link which has a node in common with i can transmit while link i is active.

2. radio-interference constraint no link that can interefere with the transmission on link i can remain active.

A feasible schedule is collision free and is defined as the set of links in G that can remain active at the same such that when a link is active none of the links in its conflict set is active at the same time.

A schedule is represented by the vector $x \in \{0,1\}^{|E|}$ where x_i is equal to 1 if link i is active and 0 otherwise. Also $x_i + x_j \leq 1$ where $i \in E$ and $\forall j \in C(i)$. The feasible schedule of a network is represented by $\mathcal{M}$.

The capacity region of a network is the set of all possible arrival rates in the network under which the queues remain stable ie the queues are of finite length. The capacity region is given by

$$\Lambda = \{\lambda | \lambda > 0 \, and \, \exists \mu \in Co(\mathcal{M}), \lambda < \mu\}$$

Co(M) is the convex hull of the set of feasible schedules in $\mathcal{M}$.

A scheduling algorithm is one which chooses a feasible schedule for the data transmission phase of every time slot. A scheduling algorithm is said to be throughput optimal or have maximum throughput when the queues remain stable for all values of arrival rates in the capacity region.

For the design of QCSMA in (Jian Bo Srikant), it is assumed that the arrival process is stochastic and the queue lengths has a Markovian description where stability refers to the positive recurrence of the Markov chain.

2.2 The Basic Scheduling algorithm

This section describes the basic scheduling algorithm used in (Jian Bo Srikant) to design QCSMA. This algorithm is based on the Gluaber dynamics from statistical physics which allows more than one link to update its transmission state in the same time slot.

Algorithm 1 Basic scheduling algorithm in time slot t (Jian Bo Srikant)

In control phase, pick a decision schedule randomly m(t) with a probability of $\alpha(m(t))$
Steps to form a transmission schedule from decision schedule
if $i \in m(t)$ **then**
    if $x_j(t-1) = 0, \, \forall j \in C(i)$ **then** then
        Turn on the transmission status with a probability p_i ie., $x_i(t) = 1$
        Turn off the transmission status with a probability $1 - p_i$ ie., $x_i(t) = 0$
    else
        Turn off the link in time slot t. ie., xi(t)=0
    end if
else
    do not change the transmission status.
end if
In the data phase, transmit according to the transmission schedule.

Every time slot, t has a control phase and a data phase. In the control phase, a set of links

are picked to be active during the data phase. This set forms the decision schedule m(t) which belongs to $\mathcal{M}_{\mathcal{O}}$ and $\mathcal{M}_{\mathcal{O}} \subset \mathcal{M}$, the feasible schedule of the network. The positive probability with which m(t) is chosen by the network is represented by $\alpha(m(t))$ and $\sum_{m(t)\in\mathcal{M}_{\mathcal{O}}} \alpha(m(t)) = 1$.

The transmission schedule is chosen from the decision schedule. For any link i $\in$ m(t), if none of the links in C(i) were active in the previous time slot, then the link i will transmit in this time slot with probability p_i and remain inactive with probability $1 - p_i$. All other links that are not part of the decision schedule will not change their transmission status.

2.3 Sensor node platform

A wireless sensor node performs the functions of sensing, processing and communicating information with other nodes in a wireless network. A mote is a type of sensor node which is small and many of them are generally spread around an area to form a wireless sensor network where they can together perform tasks such as military surveillance, monitoring conditions of soil for agriculture, locating the origin of fire for safety of a building.

Crossbows Telosb is a sensor mote (TR2420) used for this work. Telosb is an open source platform mainly used for research purposes. This platform was developed and published to the research community by UC Berkeley. This platform has long battery life as it has low power consumption and also wakes up fast from sleep state. Figure 2.1 is a block diagram of TPR2400CA (Telosb mote without the sensor suite) from (Crossbow).

Telosb mote has the following features (Crossbow).

1. USB programming and data collection capability
2. IEEE 802.15.4/Zigbee compliant RF transceiver with an onboard antenna operating at 2.4 GHz and has a data rate 250 kbps.
3. MPS430 microcontroller operating at 8 MHz with 10KB RAM
4. An external flash of 1MB for data logging
5. Sensor suite with temperature, humidity and light sensors (TPR2420)
6. Powered by two AA batteries or alternatively can plugged to a computer via USB

7. Capability to interface with other devices such as LCD displays, analog sensors and digital peripherals.

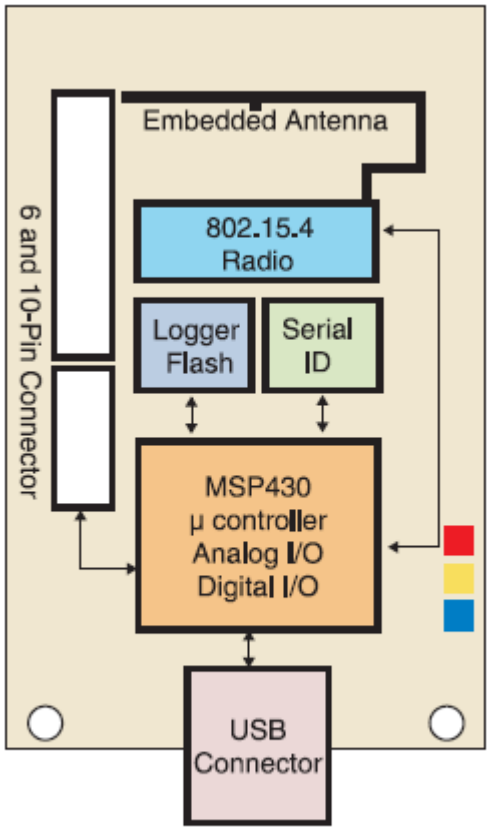


Figure 2.1 TPR2400CA block diagram from (Crossbow)

2.3.1 TinyOS (Tinyos)

The Telosb used for this work runs on the TinyOS 2.1.0 platform. TinyOS is an open source operating system for sensor networks developed by UC Berkeley(Tinyos Paper). The design of TinyOS in (Tinyos Paper) was motivated by stringent requirements of nodes in sensor networks such as limited resources, reactive concurrency, flexibility and low power. Applications on TinyOS are written in nesC language. A nesC application is a top-level configuration consisting of one or more components interacting with each other using interfaces. The interfaces can be

either a function, that is implemented by the component providing it or it can be an event, that is used by the component using the interface.

There are two broad categories of components in tinyos.

1. **Modules** provide the implementation of one or more interfaces.

2. **Configurations** connects different components throught their interfaces.

CHAPTER 3. CSMA and QCSMA Algorithms

This chapter provides details on CSMA and QCSMA algorithm and discusses the merits of the two algorithms.

3.1 CSMA Algorithm

A discrete time version of the CSMA algorithm proposed in the paper (Jian Bo Srikant) is used as a benchmark for this work. In such an algorithm, every time slot is divided into a control phase and a data phase. During the control phase, every node in the sensor network attempts to seize the channel for transmission during the data phase using a control packet. The node that wins the medium access contention during the control phase has a chance to transmit data during the data phase of that time slot.

Algorithm2 provides the steps of a discrete time version of the CSMA algorithm at a single node. During the control slot, the node that picks the lowest value for random back-off time wins the medium access contention for the data phase of that time slot. In an ideal environment when a network adopts this algorithm for its MAC protocol, all the collisions are limited to the control phase only and no collision can occur between the data packets during the data phase.

3.2 Design of QCSMA Algorithm

Before we begin the discussion on QCSMA algorithm, we need to have some background on the general max weight scheduling algorithms.

Algorithm 2 Discrete time version of CSMA algorithm at every node (Jian Bo Srikant)

In control slot, randomly pick a back-off time and sense the channel for transmissions.
if channel is busy **then**
    stop the back-off timer
    turn off the transmission status and quit sensing the channel.
else
    continue to transmit the control packet when the back-off timer expires
    continue channel sensing
end if
if Backoff timer expires **then**
    if control message collides **then**
        turn off the transmission status and quit sensing the channel.
    else
        turn on the transmission status and quit sensing.
    end if
end if
In data phase, transmit a data message if the transmission status is on else do not transmit any data message.

3.2.1 General max weight scheduling algorithms(Srikant Lei)

Max weight scheduling for multi-hop radio networks was first introduced in (Tassiulas Ephremides) and shown to be throughput-optimal. The throughput-optimality property of a scheduling algorithm was introduced in (Tassiulas Ephremides). Here we discuss the general max weight algorithm from (Srikant Lei) which uses queue lengths as link weights so that if a flow is not serviced then the size of the queue builds up large enough that its link weight is increased and hence receives higher priority to be serviced. MWS can be represented by

$$\mathcal{M}(t) \in arg \max_{M(h)} \sum_{l} w_l(q_l(t)) \mathcal{M}_l^{(h)}$$

source: (Srikant Lei)

$\mathcal{M}(t)$ is Max weight schedule scheduled by the network at time t. $q_l(t)$ is the length of the queue at time t and w_l is the weight of link at time t. MWS can achieve low delays but cannot be implemented efficiently on distributed nodes with limited computing capabilities.

3.2.2 QCSMA Algorithm(Jian Bo Srikant)

QCSMA approximates Max weight scheduling in a distributed manner(Srikant Lei). 3 provides the QCSMA protocol at each link in a given time slot. In every time slot a link decides to include itself in set D with a certain probability. If there are no neighbors contending to transmit in this slot, the link becomes part of set D. During the data phase of a given time slot, all the links that are part of setD and those that had no neighbors transmitting in the previous time slot, transmit data with a link activation probability which is a function of queue length. In (Jian Bo Srikant; Srikant Lei) it is proved that such a QCSMA has a Markov chain that is irreducible and aperiodic due to the following reasons.

1. D is an independent set as no two of the links present in D conflict with each other.

2. From any state x, the QCSMA process guarantees that the transition probability from x to the schedule in which all links are strictly positive.

Hence QCSMA can keep the queue lengths stable for all stochastically varying arrival probabilities in the capacity region of the network.

Algorithm 3 Distributed implementation of QCSMA algorithm in time slot t (Srikant Lei)

Calculate the weight for the link as $w_l = log(1 + q_l(t))$
In the control phase transmit a control message with probability $\beta(0 < \beta < 1)$ (fixed for every link), independent of all other links.
if control message collides with that of a neighbor's **then**
    The link does not become part of decision set D
else
    the link becomes a part of D
end if
for all l∈D **do**
    if link l does not have a neighbor in the conflict graph that was transmitting in the previous time slot then
    link l turns on the transmission with probability $\frac{e}{1+e}$ and turns off the transmission with $\frac{1}{1+e}$
end for
for the other links l∉D, do not change the transmission status of the link.

CHAPTER 4. CSMA QCSMA Implementation on TelosB

This chapter presents the implementation details for the discrete time version of CSMA and QCSMA on Telosb platform.

4.1 TinyOS Components (Tinyos Documentation)

Below is a brief description from (Tinyos Documentation) of the tinyos components used for the implementation of CSMA and QCSMA.

4.1.1 Main C

Provides the system interface to boot TinyOS. It wires the boot sequence implementation to the scheduler and hardware resources.

4.1.2 LedsC

This component provides the interface Leds through which a component that uses this interface can program the leds on the device. Toggling specific leds in certain loops of the program is useful while debugging the code on hardware.

4.1.3 TimerMilliC( )

Provides the interface Timer which takes a time value in milli seconds and fires an event to the component that uses this interface. The statements to be executed when the event is fired should be implemented in the component that uses the interface. TimerMilliC is a generic component. In order to implement different timers in the design different instances of TimerMilliC needs to be created and wired to separate timer interfaces in the configuration of the main application.

4.1.4 CC2420ActiveMessageC

This component provides interface to access the CC2420 radio on the hardware. The component connected to it needs to have a separate interface for every message (based on AM id). Acknowledgements are received by the component connected to it based on the destination address of the sent packet and group (AM id).

4.1.5 QueueC

Component that provides interface to create a queue in the memory and perform functions on it such as push, pop, provides the size of the queue and flags an error if an attempt is made to push a data member in a full queue. The component that uses this interface needs to specify the size of the queue that is required and the size of the data. For the implementation of CSMA and QCSMA, a queue length of 25 and data size of 16 bits was created at every node.

4.1.6 RandomC

This component generates a random 16 or 32 bit integer. A component that is connected to it requests RandomC component by executing a command call rand16() or rand32(). This is helpful for generating probabilities required in the implementation of CSMA and QCSMA.

4.1.7 HplCC2420PinsC

This component provides access to the I/O pins for CC2420 radio connected to a TI MSP430 processor. In this implementation, Clear Channel Assessment CCA pin is used for sensing if the channel is busy or not.

NetworkNodeCSMAC and NetworkNodeQCSMAC are the components implementing the CSMA and QCSMA scheduling algorithms respectively. They use interfaces provided by the TinyOS components described above.

4.2 Slotted CSMA implementation

Slotted CSMA on Telosb is used as a reference implementation. The implementation details are provided below in terms of actions performed at every node when a particular event occurs. The six main events for a CSMA node are successful boot event, end of data period event (Timer2 expires), end of control period event (Timer3 expires), end of back-off period (Timer4 expires) and event when a node detects transmission of neighbors control packet.

Algorithm 4 Distributed slotted CSMA

```
1: if Event (on successful boot) then
2:     Synchronize all nodes according to the synchronization algorithm (Algorithm-1)
3:     Schedule Timer1 to expire after control period (end of control phase)
4:     Set variable win = TRUE
5: end if
6: if Event (Timer2 expires/end of data period) then
7:     Schedule Timer1 to expire after control period.
8:     Generate data payload with arrival probability
9:     Select a back-off time randomly and set Timer4 to expire after that time interval.
10: end if
11: if Event (Node detects a control packet being transmitted by its neighbor) then
12:     if node has not yet transmitted its own control packet then
13:         set win=FALSE
14:     elsestop Timer4 if it is still running as the node should not transmit a control packet.
15:     end if
16: end if
17: if Event (Timer1 expires) then
18:     Schedule Timer4 to expire after data period.
19:     if win= TRUE then
20:         transmit the data message
21:     else
22:         do nothing
23:         Set win=TRUE (ready for contention for the next time slot).
24:     end if
25: end if
26: if Event (Timer4 expires) then
27:     Transmit control packet
28:     if control packet is sent successfully then
29:         set win=TRUE
30:     else
31:         win=FALSE
32:     end if
33: end if
```

4.3 QCSMA implementation

QCSMA implementation on Telosb nodes also uses expiring events of Timer1 and Timer2 to signify the end of control phase and the end of date phase respectively. The state of the neighboring links in the previous time slot is saved in the variable neightrans_prev. Packets are generated every time slot with arrival probability of α and placed in the queue.

Algorithm 5 Distributed QCSMA implementation

if Event (On successful boot) **then**
    Synchronize with respect to the synchronization algorithm (Algorithm-1)
    Schedule Timer1 to expire after control period.
end if
if Event (Every time slot after synchronization) **then**
    Generate a data packet and place it in the queue with arrival probability
    Place the packet on the queue if it is not full
    else increment count_drop which keeps track of the number of packets dropped
end if
if Event (Timer2 expires) **then**
    Save the state of neighbor transmissions in the previous time slot (neightrans) in neightrans_prev.
    neightrans = FALSE (no neighbor has transmitted a data packet in the new time slot).
    setD = TRUE (make the node part of the decision set in the new time slot) with a probability of 0.95.
    if queue_size! =0 **then**
        pick a random time to transmit control packet in the control phase and set Timer4 to expire after that time
    elseset variable setD=FALSE;
    end if
    Set Timer1 to expire after control period.
end if
if Event (Node detects a control packet being transmitted by its neighbor) **then**
    if node is yet to transmit a control packet in this time slot **then**
        cancel transmission of control packet and setD=FALSE
    end if
end if
if Event (Node detects a data packet being transmitted by its neighbor) **then**
    Set variable neightrans TRUE
end if

28: **if** Event (Timer4 expires) **then**
29: **if** node has not detected any control packet from the neighboring nodes **then**
30: transmit a control packet
31: **if** control packet transmssion was unsuccessful **then**
32: set variable setD=FALSE
33: **end if**
34: **end if**
35: **end if**
36: **if** Event (Timer1 expires) **then**
37: **if** setD=TRUE and neightrans_prev=FALSE **then**
38: turn on the transmission state of the node with a probability of $\frac{e^{wl}}{1+e^{wl}}$ and turn off the transmission state of the node with a probability of $\frac{1}{(1+e^{wl})}$
39: **else**
40: do not change the transmission state of the node.
41: **end if**
42: Set Timer2 to expire after data period.
43: **if** transmission state of the node is on **then**
44: transmit data packet
45: **else**
46: do nothing.
47: **end if**
48: **end if**

4.4 Radio Communication in TinyOS

TinyOS 2.x has an abstract data type or message buffer called message_t (Tinyos TEP). The structure of message_t defined in TinyOS 2.x is as given below. From (Tinyos TEP), we have the details on the different structures within message_t as follows.

```
typedef nx_struct message_t {
  nx_uint8_t header[sizeof(message_header_t)];
  nx_uint8_t data[TOSH_DATA_LENGTH];
  nx_uint8_t footer[sizeof(message_footer_t)];
  nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

The message_header is the header of the message which contains information on clock frequency, destination address, am_id type, length of the payload. The message_data contains the payload for single-hop packet. Its maxium length is TOSH_DATA_LENGTH. The default

value of this is 28 bytes. TinyOS application can redefine the value during compile time. If there are 2 applications with different TOSH_DATA_LENGTH communicating with each other, packets may be dropped as the packet layer would drop any packet that has a length lesser than TOSH_DATA_LENGTH. The message_footer field ensures that the buffer has enough space to store the footers for all underlying link layers. Finally, message_metadata stores data such as transmission power, (Received Signal Strength Index) RSSI, (Cyclical Redundancy Check) CRC, acknowledgment status and time which are not transmitted but are used and collected by single-hop stacks.

The message buffer message_t is of fixed length for a given platform and this is especially useful while passing messages between different link layers. For instance, when a packet is received by a node it can be placed in the memory and a pointer to that can be passed between the CC2420 radio stack and TinyOS serial stack. Every link layer needs to have its own structure of message_header_t, message_footer_t and message_metadata_t defined and the corresponding structures on the platform must be a superset of all the fields defined in its link layers.

4.4.1 Active Message Layer (AM Layer)(Tinyos Documentation)

Multiple radio messaging services are possible using the same radio on a platform by means of Active Message Layer. It uses an identifier called AM type to different between the multiple services using the radio for communication.

4.4.2 AM Layer Interfaces (Tinyos Documentation)

AMPacket interface for accessing message_t abstract data type, getting pointer to its payload, providing AM local address, single-hop AM destination address, AM group and other functionalities to the query packet. AMSend interface used for sending an AMPacket, receiving acknowledgement, canceling a pending message and other functionalities for sending an AM Packet. It sends the packet to the node with an AM address set as destination address. AMReceive interface used for receiving an AMPacket, getting a pointer to its payload area and other functionalities for reception of an AMPacket.

TinyOS AM components implement the AM interfaces. Some of the AM Components used in this work are AMSenderC and AMReceiver which are components for sending, receiving AM messages. These are generic components and need to be instantiated separately for every am_id.

The structure used for the definition of control packet and data packet are given below. The ctrl packet is 9 bytes long and the data packet is 13 bytes long.

```
typedef nx_struct BlinkToRadioCtrlMsg{
        nx_uint16_t nodeid; //node identifier
        //flag used during synchronization (explained in the next chapter)
        nx_uint8_t flag;
        //The count for the control msg. Same as the time slot number.
        nx_uint16_t count;
        //time of Arrival of the packet. Used by the receiving node
        nx_uint32_t timeofArrival;
}BlinkToRadioCtrlMsg;
typedef nx_struct BlinkToRadioMsg{
        nx_uint16_t nodeid;
        nx_uint32_t timeofArrival;
        nx_uint8_t size; //queue size
        nx_uint16_t no; // 16-bit payload
        //the number of packets generated
        nx_uint16_t totalpackets;
        //number of packet generation cycles
        nx_uint16_t totalcount;
}BlinkToRadioMsg;
```

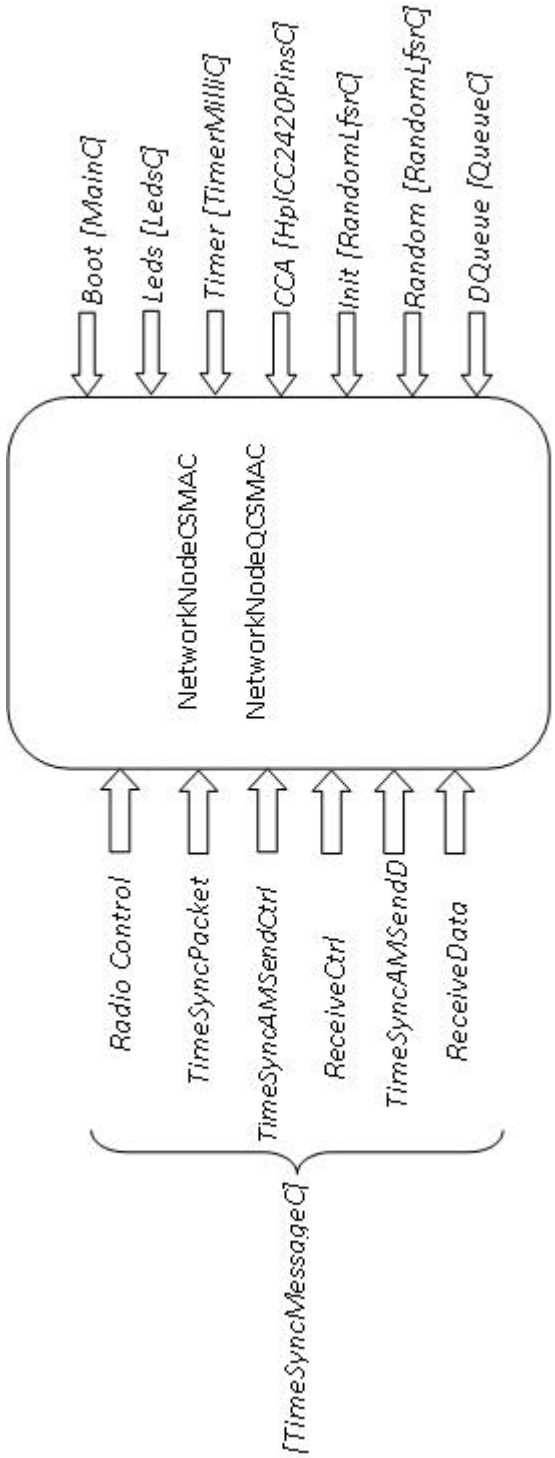


Figure 4.1 Block diagram showing interfaces used by CSMA and QCSMA components

CHAPTER 5. Line Network Setup

This chapter describes the methodology used for construction of a line network within which the performance of the two algorithms are analyzed.

5.1 Varying Transmission Power

Transmission radius for a wireless transmitter node is the radius around it where the (Received Signal Strength Index) RSSI strength is strong enough for any packet transmission to detected by a wireless receiver node. The RSSI strength of a radio signal varies directly with the transmission power of the transmitter node.

At full transmission power the transmission radius for a telosb node can be upto 30m. In order to be able to fit the entire line network into a single room, the transmission power was lowered for the telosb motes in the line network. The transmission power for a telosb mote can be varied between -25dBm and 0dBm by passing values between 1 to 31 to DCC2420_DEF_RFPOWER at compile time.

5.2 Initial Synchronization

For the CSMA and QCSMA algorithms described in chapter 4 it was assumed that the network nodes had their time slots synchronized. In our line network we use the simple synchronization algorithm below for both CSMA and QCSMA cases in order to synchronize the time slots of neighboring nodes in the network.

The nodes of the network during synchronization can be in one of the 3 states.

1. Leader node - transmits a control packet before any of its neighbors. Neighboring nodes try to synchronize their time slots with the leader node through the control packets

received from the leader node.

2. Follower node synchronizes its time slots with the leader node. Once synchronized with the leader node, transmits a sub-control packet so that its neighboring nodes who are not in the transmission range of the Leader node can synchronize to the leader node.

3. Sub-follower node is a follower of the follower node and synchronizes its time slots with respect to that of the follower node through the sub-control packets transmitted by the follower node.

Algorithm 6 Simple Synchronization for network nodes in the line network

```
1: if Event (on successful boot) then
2:     transmit a control packet for synchronization if the node does not hear any control packet or sub-control packet from its neighbors.
3: end if
4: if Event (Timer2 expires) then
5:     if node is a leader node then
6:         transmit a control packet
7:     end if
8:     if node is a follower node then
9:         if follower node is not yet synchronized with the leader node then
10:            sense the CCA pin at the beginning of the control period
11:            if channel is busy then
12:                the leader node is sending a control packet and the time slot of the node is synchronized with that of the leader node.
13:            else
14:                continue synchronizing with the leader node by sensing the CCA pin every-time Timer2 expires.
15:            end if
16:        else
17:            send a sub-control packet after 500ms.
18:        end if
19:    end if
20:    if node is a sub-follower node then
21:        set Timer3 to expire after 500ms.
22:    end if
23: end if
```

```
24: if Event (receives a control packet from its neighbor) then
25:     if control packet received after it transmits its own control packet then
26:         ignore, do nothing.
27:     elserestart the timers to synchronize beginning of the control slot with the neighbor
    (leader node) whose control packet was received.
28:     end if
29: end if
30: if Event (Timer3 expires) then
31:     if node is sub-follower then
32:         sense the channel through the CCA pin
33:         if channel is busy then
34:             node is synchronized with respect to the sub-leader
35:         end if
36:     end if
37: end if
38: if Event (node receives a sub-control packet) then
39:     if node has not picked a sub-leader then
40:         restart the timers to synchronize with the node that sent the sub-control packet
41:     end if
42: end if
```

CHAPTER 6. Results

This chapter discusses the results of the experiments that were conducted to compare the performance of QCSMA vs slotted version of CSMA.

6.1 Experiment results of slotted CSMA and QCSMA

The MAC protocols were studied under the following three different conditions of packet arrival probabilities at the nodes : sum of arrival probabilities at link nodes is lesser than 1 , almost equal to 1 and greater than 1. As the queue size is finite at every node, the packets are dropped from the queue if the rate at which the queue is serviced or the rate at which packets are transmitted from the queue is lesser than the rate at which the packets arrive at the queue. The throughput and delay of the network can be studied by observing the pattern of the queue lengths and the packet drop variation at the nodes as time progresses. For a scheduling algorithm with high throughput the queue lengths must be stable for large values of arrival probabilities. The delay experienced by packets in the queue is a measure of the delay performance of a scheduling algorithm. (Jian Bo Srikant).

6.1.1 Varying the node arrival probabilities for a fixed queue length

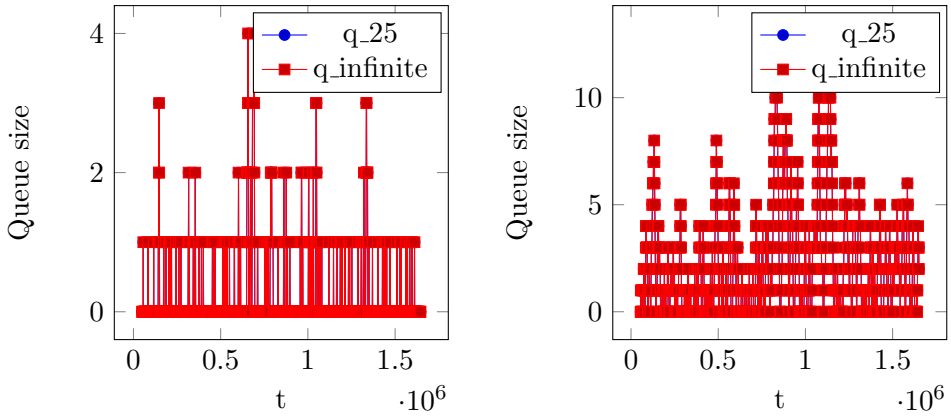
The maximum queue size at each node in the line network was set to 25 as this queue size was sufficient to study the behaviour of the two MAC protocols. Figures 6.1 - 6.12 show the actual queue size (shown as q_25) and the infinite queue size (shown as q_infinite) for the six cases at the nodes 2 and 3 in the line network for CSMA and QCSMA. The infinite queue size at a node is the sum of the actual queue size and the packets dropped from the queue due to overflow. For QCSMA, q_infinite is used for calculating the link activation probabilities at the

Case	Arrival probabilities of Nodes 1,2,3,4
Case1	0.2, 0.4, 0.2, 0.4
Case2	0.4, 0.2, 0.3, 0.1
Case3	0.3, 0.6, 0.3, 0.6
Case4	0.6, 0.3, 0.7, 0.2
Case5	0.6, 0.7, 0.5, 0.8
Case6	0.5, 0.6, 0.5, 0.6

Table 6.1 Arrival probability values for different cases

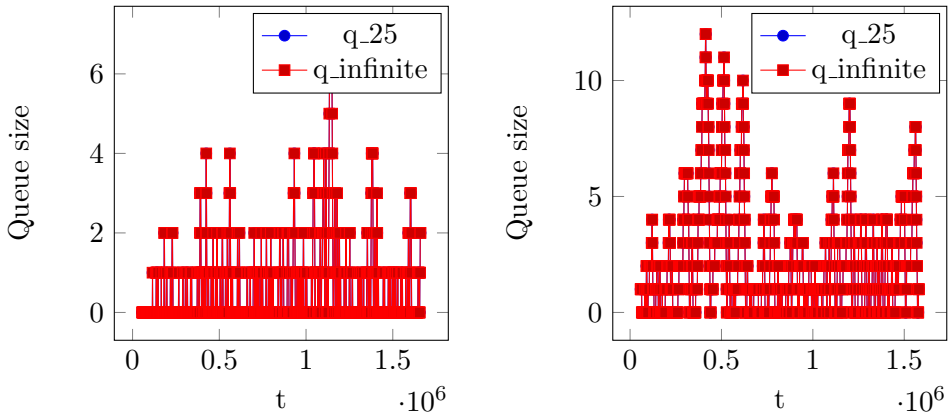
individual nodes. The neighbor's arrival probabilities are set symmetrical for cases 1,6,9 and unsymmetrical for cases 2,4,5.

For Case1 and Case2 where sum of the arrival probabilities of two neighboring nodes is less than 1, figures 6.1- 6.4 show that both CSMA and QCSMA protocols maintain the queue sizes lower than the max queue size at all times. Overall, CSMA showed lower queue sizes compared to QCSMA in case1 at both nodes and in case2 at node3. As sum the arrival probabilities at the link nodes was increased and when their sum was almost equal to 1 and greater than 1, some packets were dropped in both CSMA and QCSMA (except for QCSMA Case3 at Node3) as the packets arrived at the links at a rate higher than they could be transmitted on the network. For Case3, the packets dropped were lower for QCSMA compared to CSMA at Node2 and no packets were dropped at QCSMA at Node3 whereas in CSMA 17 packets were dropped at the end of 1700 seconds (refer Figure 6.6). Case4 results show that more packets were dropped in QCSMA compared to CSMA at Node2 ($\alpha_2 = 0.3$) and the rate at which the queues build up initially is higher for QCSMA at Node3 compared to CSMA and the queue sizes oscillate uniformly bewteen 25 and a lower value thereafter. In case of CSMA, although no packets are dropped initially, there is sudden surge in q_infinite higher than that of QCSMA after 1400 seconds (refer figure 6.8 a) and packets are dropped. The packets dropped and the rate at which the queue sizes varied were comparable for both protocols in Case5 at both nodes. For Case6, the packets dropped at the Nodes were remarkably lower in case of QCSMA for both nodes 2 and 3 compared to that of CSMA as seen from figures 6.4 - 6.5.



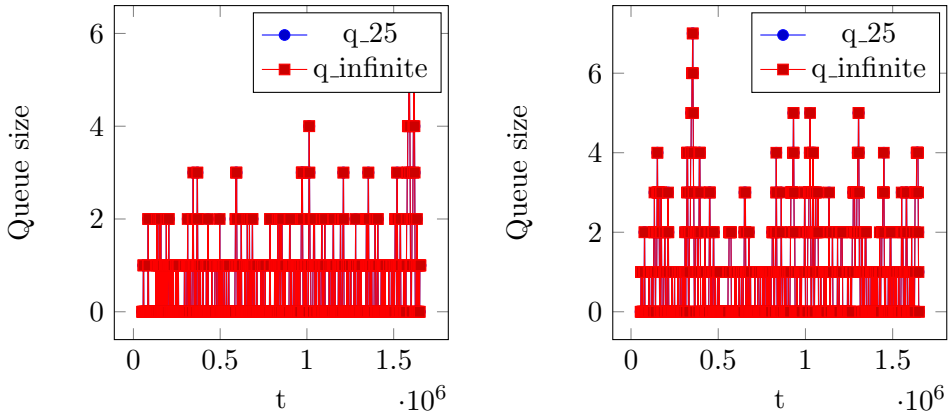
(a) CSMA Case1 node2 (b) QCSMA Case1 node2

Figure 6.1 CSMA-QCSMA comparision Case1 at Node2



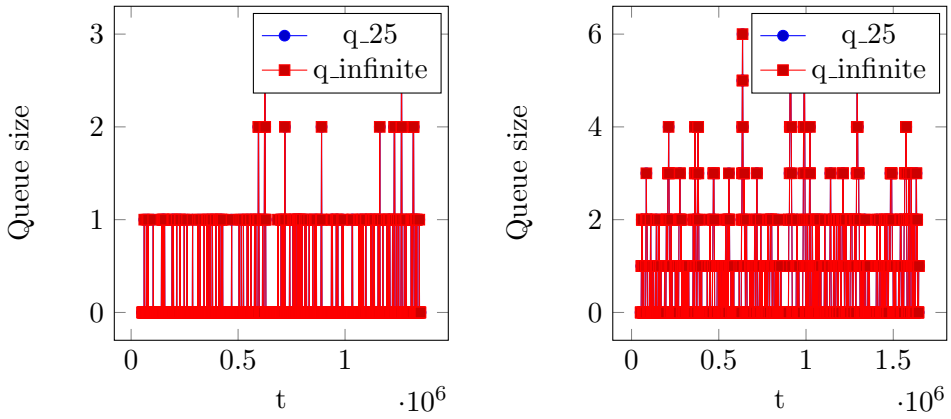
(a) CSMA Case1 node3 (b) QCSMA Case1 node3

Figure 6.2 CSMA-QCSMA comparision Case1 at Node3



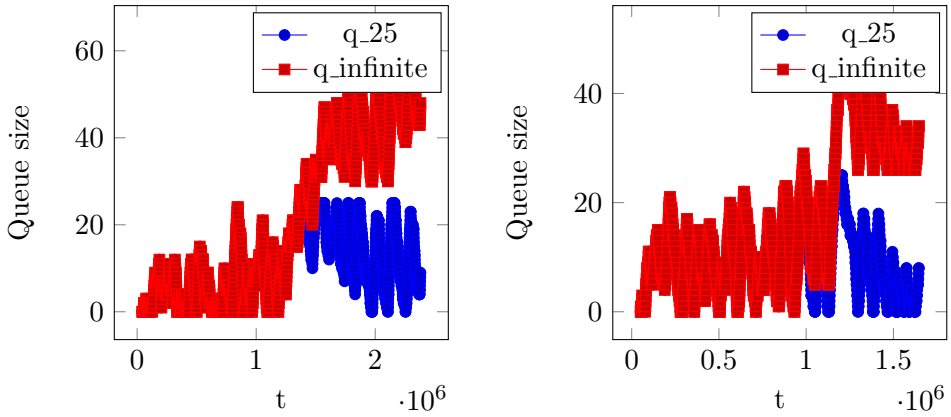
(a) CSMA Case2 node2 (b) QCSMA Case2 node2

Figure 6.3 CSMA-QCSMA comparision Case2 at Node2



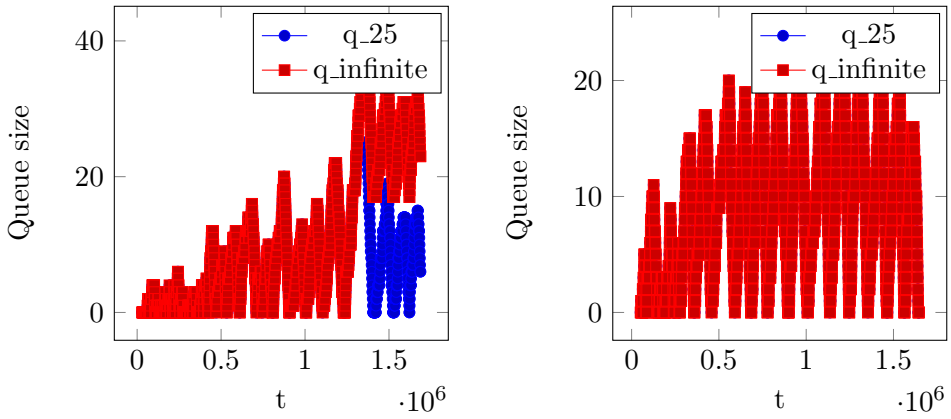
(a) CSMA Case2 node3 (b) QCSMA Case2 node3

Figure 6.4 CSMA-QCSMA comparision Case2 at Node3



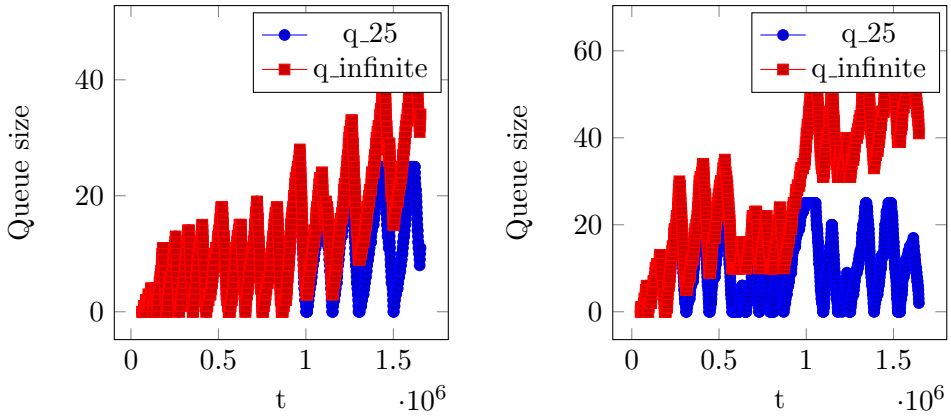
(a) CSMA Case3 node2 (b) QCSMA Case3 node2

Figure 6.5 CSMA-QCSMA comparision Case3 at Node2



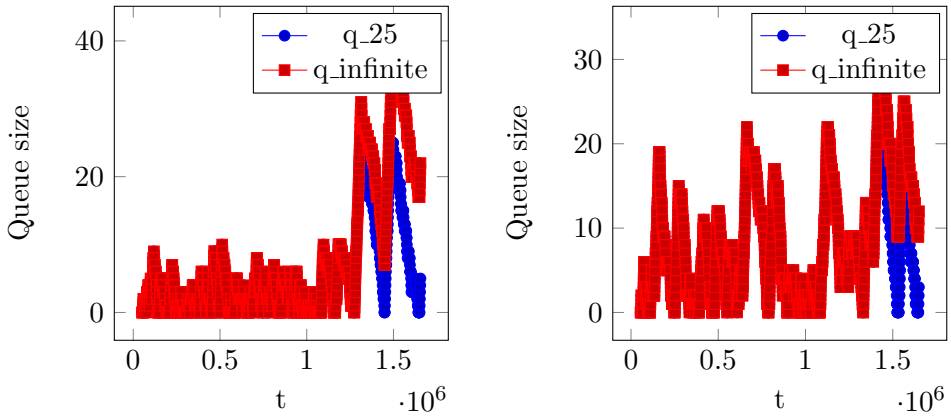
(a) CSMA Case3 node3 (b) QCSMA Case3 node3

Figure 6.6 CSMA-QCSMA comparision Case3 at Node3



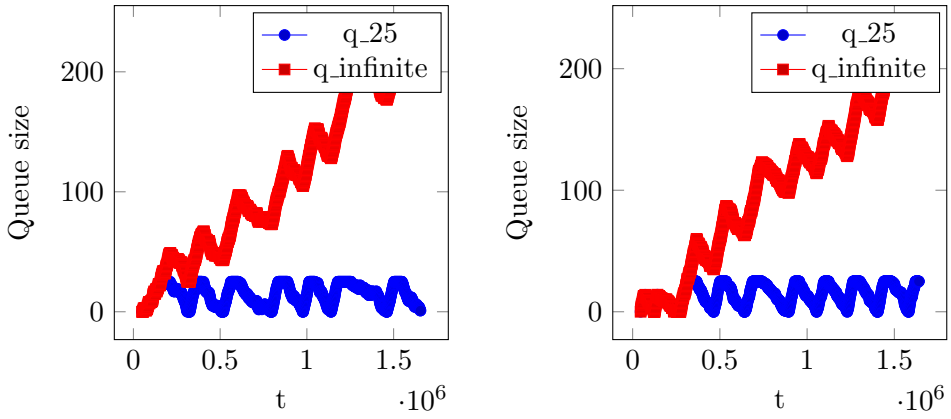
(a) CSMA Case4 node2 (b) QCSMA Case4 node2

Figure 6.7 CSMA-QCSMA comparision Case4 at Node2



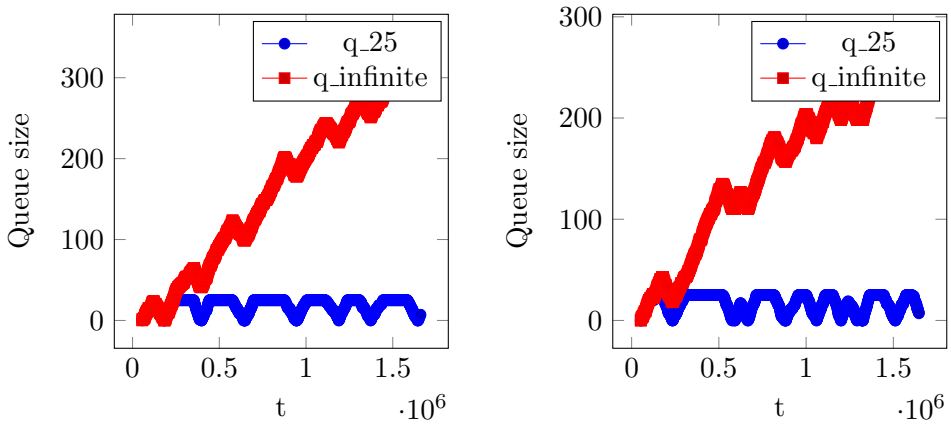
(a) CSMA Case4 node3 (b) QCSMA Case4 node3

Figure 6.8 CSMA-QCSMA comparision Case4 at Node3



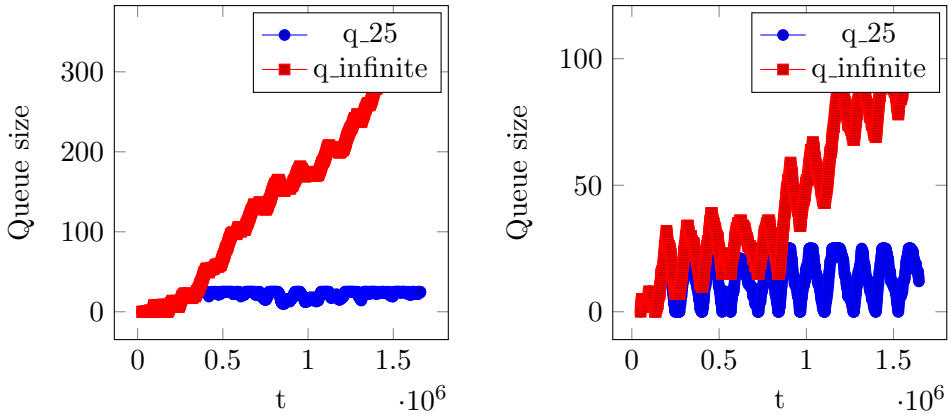
(a) CSMA Case5 node2 (b) QCSMA Case5 node2

Figure 6.9 CSMA-QCSMA comparision Case5 at Node2



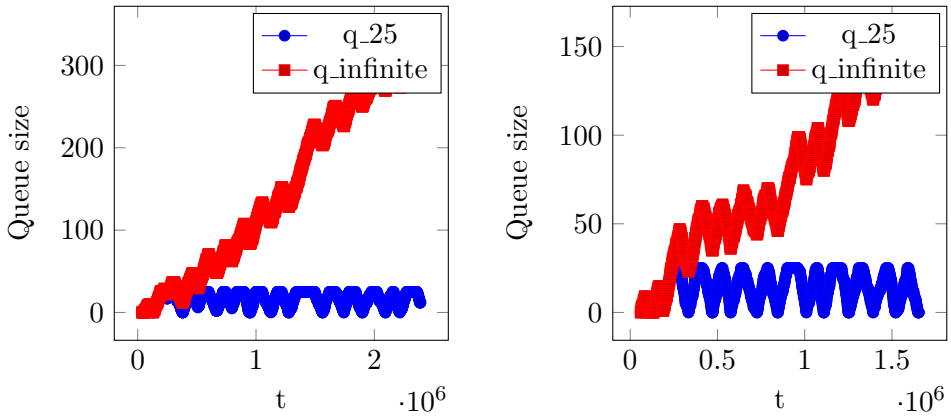
(a) CSMA Case5 node3 (b) QCSMA Case5 node3

Figure 6.10 CSMA-QCSMA comparision Case5 at Node3



(a) CSMA Case6 node2 (b) QCSMA Case6 node2

Figure 6.11 CSMA-QCSMA comparision Case6 at Node2



(a) CSMA Case6 node3 (b) QCSMA Case6 node3

Figure 6.12 CSMA-QCSMA comparision Case6 at Node3

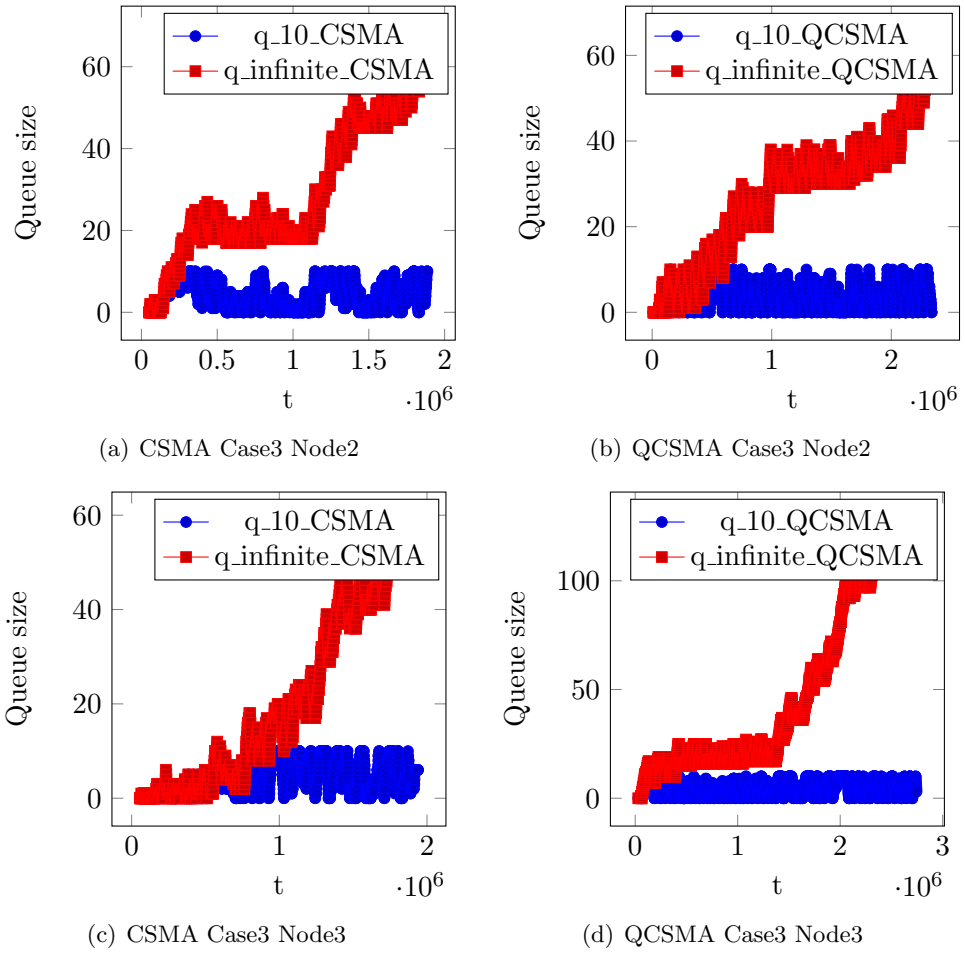
6.1.2 Varying the Maximum Queue Size Value

For these set of experiments the maximum value of queue size (Q_max) was changed from 25 to a lower value, 10 and a higher value, 50 and experiments were conducted for cases 3 and 6. Larger number of packets were dropped for case 3 and case 6 at Q_max=10 and the queue sizes increased linerarly with time indicating that the queues were unstable even when the sum of the arrival probabilities were almost 1 unlike the case when Q_max=25. And for Q_max=50, no packets were dropped for case3 unlike the packet drops observed in Q_max=10 and 25 for case3 as the queues were large enough to accomodate for the increased arrival rate of the packets. For both cases 3 and 6 when Q_max=10 and Q_max=50, the total packets dropped in case of QCSMA was lower compared to that of CSMA as observed from figures 6.13 - 6.16.

6.2 Need for modification of the QCSMA implementation

In QCSMA when a node transmits a control packet before its neighbors, it is included in setD and its neighbors, on receiving the control packet do not attempt to sieze the channel during this time slot. In a non-ideal environment, where the link quality vary with time due to channel fading, packets could get lost during the transmission. Suppose a control packet transmitted by a node is lost due to channel fading, the neighbors do not receive the control packet from the node and hence could transmit its own control packet despite not being the first among its neighbors to transmit a control packet. If the link activation probabilities are satisfied for both the nodes then both the node and its neighbor can turn on resulting in data collision between conflicting neighbors violating the protocol. Also, it is possible that the data packets transmitted by a neighboring node can get lost in such a non-ideal environment and incorrect previous data transmission status of the neighbor would be used by a node when it is determining whether to include itself into setD or not.

Figure 6.6 show the finite queue sizes for the nodes 1,2 and 3 for the QCSMA experiment run for the cases 1,3 and 6. Since the neighboring nodes in the line network conflict with each other for data transmission in any time slot the queue sizes for the conflicting neighboring



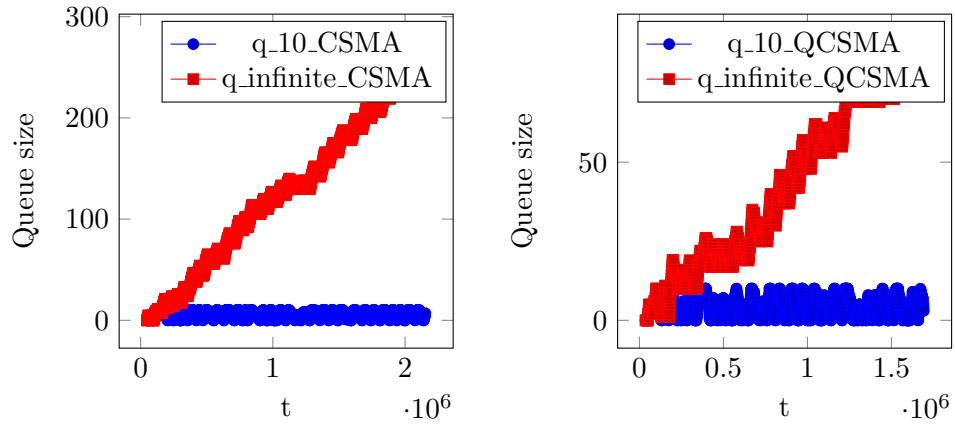
(a) CSMA Case3 Node2

(b) QCSMA Case3 Node2

(c) CSMA Case3 Node3

(d) QCSMA Case3 Node3

Figure 6.13 CSMA-QCSMA max queue size=10 Case3



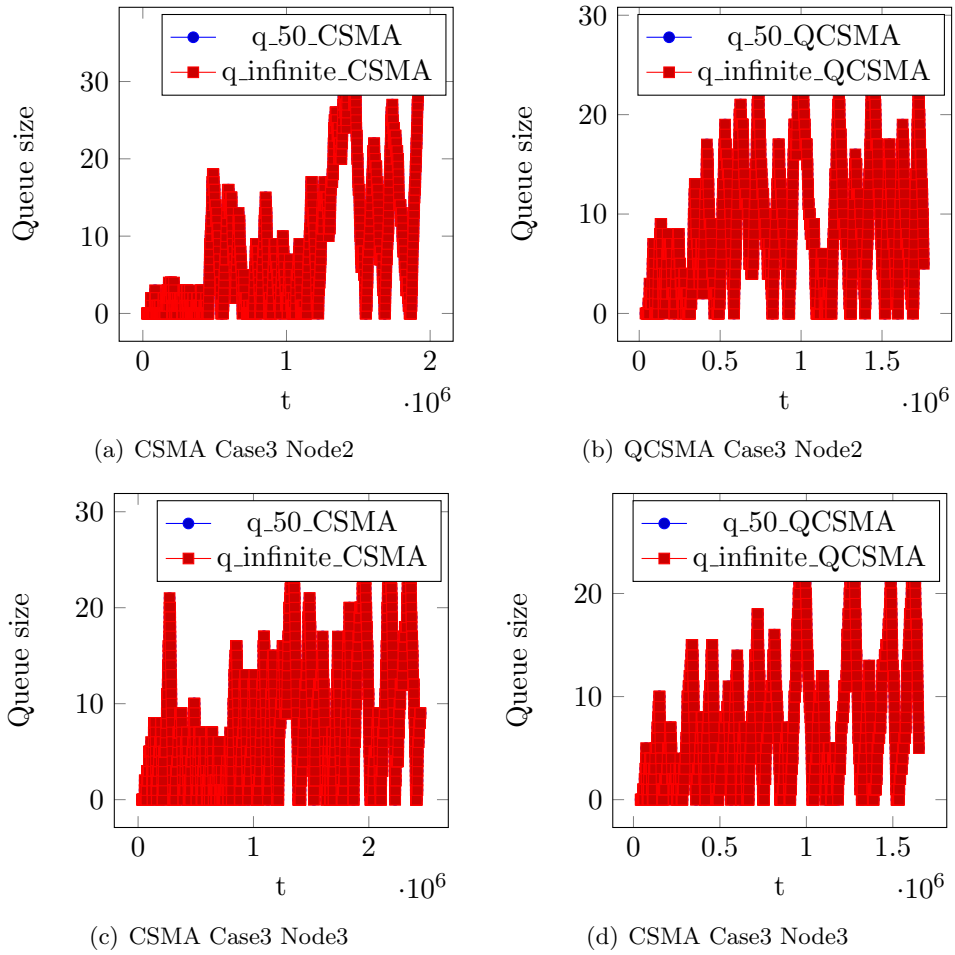
(a) CSMA Case6 Node2

(b) QCSMA Case6 Node2

(c) CSMA Case6 Node3

(d) QCSMA Case6 Node3

Figure 6.14 CSMA-QCSMA max queue size=10 Case6



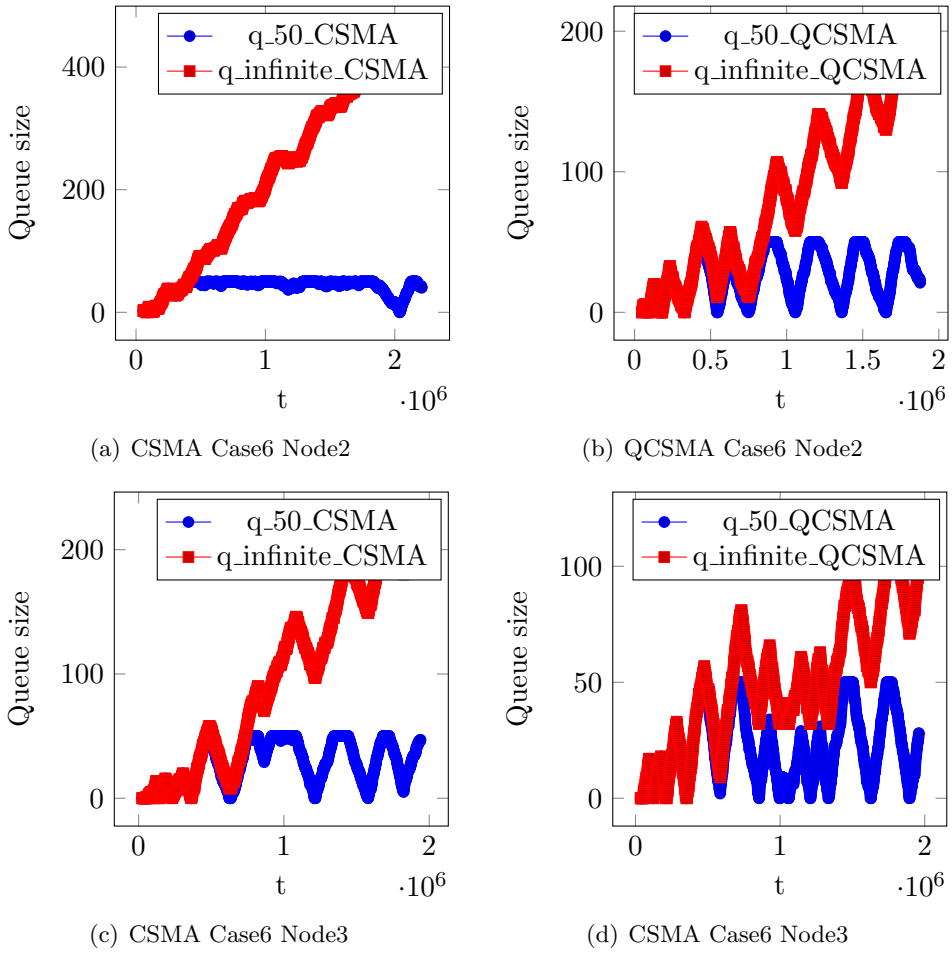
(a) CSMA Case3 Node2

(b) QCSMA Case3 Node2

(c) CSMA Case3 Node3

(d) CSMA Case3 Node3

Figure 6.15 CSMA-QCSMA max queue size=50 Case3



(a) CSMA Case6 Node2

(b) QCSMA Case6 Node2

(c) CSMA Case6 Node3

(d) CSMA Case6 Node3

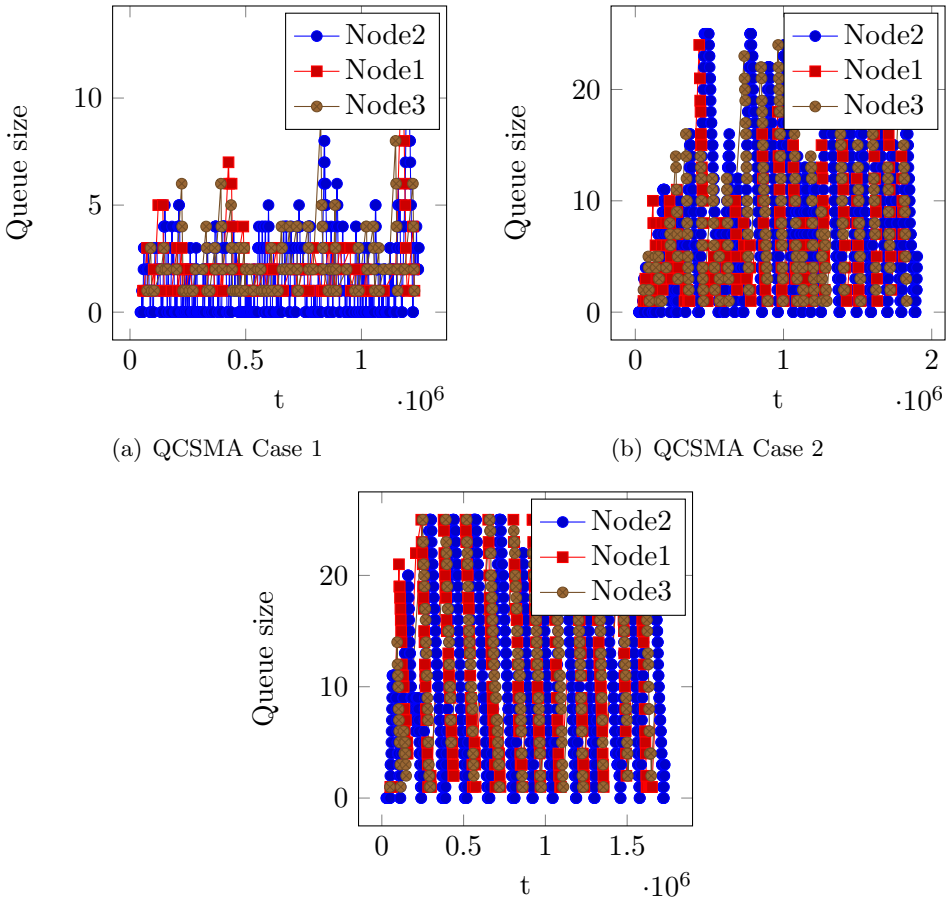
Figure 6.16 CSMA-QCSMA max queue size=50 Case6

nodes cannot decrease at the same time. It is readily observable in Figure 6.6 that the queue sizes for the neighboring nodes decrease at the same time which should not be the case for ideal QCSMA implementation.

In order to investigate this scenario of data collisions between conflicting neighboring nodes, the control and data packets were numbered at each node and this information was included in the packets that were transmitted by every node. The control and data messages received by a node were logged during the experiment run. On analysing these log files it was found that in the time slots just before the simultaneous data transmissions of two conflicting neighbors occurred, the numbers on the data and control packets were not contiguous. In other words, in original QCSMA algorithm there was no handling of lost control and data packets which led to the conflicting neighbors' queue sizes to be dropped simultaneously in a non-ideal environment where packets were lost.

For instance, in time slot x, only node2 transmitted a control packet and no data packets were transmitted by either node2 or its neighbors. In the next time slot x+1, node3 transmitted the control packet first but it was lost and was not received by node2, both node2 and 3 transmit their control packets individually and turn their transmission status to on when appropriate conditions are satisfied. This results in simultaneous data transmission in time slot x+1. In time slot x+2 since both nodes 2 and 3 heard data transmission from its neighbors, they do not change their transmission status which is on and this continues for several consecutive time slots until the queue sizes decrease to a point where the link activation probabilities are not satisfied.

Similar to QCSMA, the packets were also lost for the CSMA case but the effect was not compounded in the consecutive time slots like QCSMA as the link activation decision was made independently in every time slot without any information on the transmission status of its neighbors in the previous time slots.



(a) QCSMA Case 1

(b) QCSMA Case 2

(c) QCSMA Case 3

Figure 6.17 QCSMA comparing conflicting queue sizes for QCSMA

6.2.1 Modified QCSMA implementation

In order to reduce the effect of the lost data and control packets on the queue sizes, the original QCSMA was modified according to 7. The transmission history for the node and its neighbors in the previous time slot is recorded in the variables (transmit_prev) and (neigh-trans_prev). In the modified QCSMA implementation, when a control packet is lost among two neighboring nodes and both nodes end up transmitting data during the data phase then the nodes set the variables (transmit_prev) and (neightrans_prev) to TRUE. In the next time slot they turn off their transmitters as both (transmit_prev) and (neightrans_prev) is set to TRUE. This prevents the compounded data collision effect that we observed in case of original implementation of QCSMA.

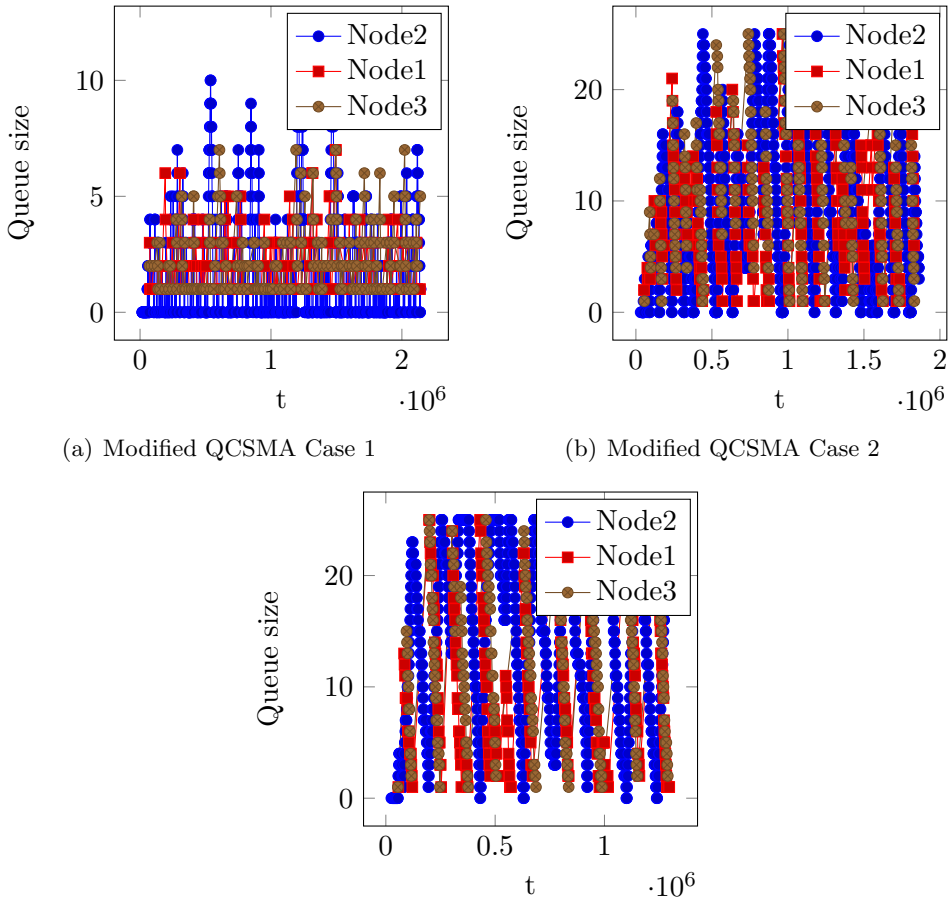
Algorithm 7 QCSMA implementation modification

1: **if** neightrans_prev is TRUE and transmit_prev is TRUE **then**
2: Turn off the transmission status
3: **end if**
4: In the current time slot store the current transmission status (transmit_prev) and neighbor's transmission status (neightrans_prev) to be used in the next time slot.

6.2.2 Experiment results for the modified QCSMA

Figure 6.7 show the relative queue sizes for the conflicting neighbors in the line network for modified QCSMA for cases1, 2 and 3. As seen for the graphs the queue sizes are linked as should be the case for ideal QCSMA and when one queue size increases the other decreases.

Figures 6.15-6.20 show the queue sizes q_25 and q_infinite for modified QCSMA at nodes 2 and 3 of the line network under different cases. Modifying the QCSMA implementation resulted in almost same throughput for cases1,2 (at both nodes 2 and 3), case3 (at node2). Packets dropped increased for case3 at node3 in the modified QCSMA protocol comparable to that of CSMA case. The packets dropped decreased for case4 at node2 in the modified QCSMA and was comparable to that of CSMA whereas no change was observed in the number of packets dropped in case4 case at Node3. The queues are unstable similar to that of CSMA and original QCSMA for cases 5 and 6.

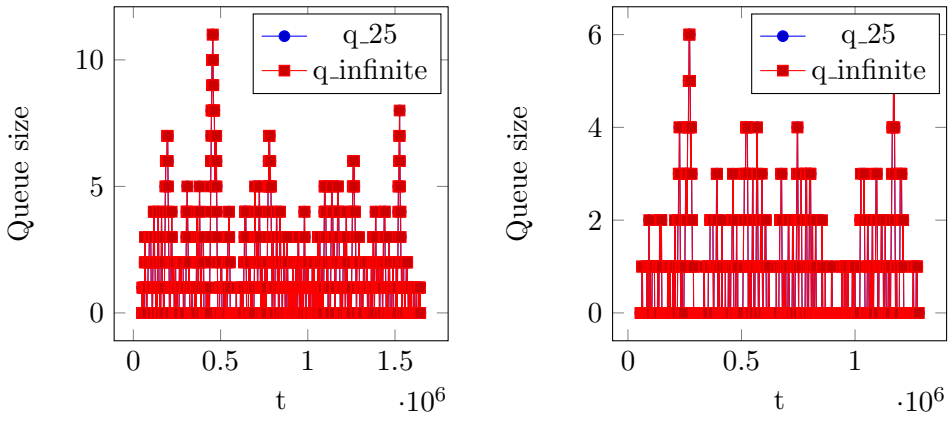


(a) Modified QCSMA Case 1

(b) Modified QCSMA Case 2

(c) Modified QCSMA Case 3

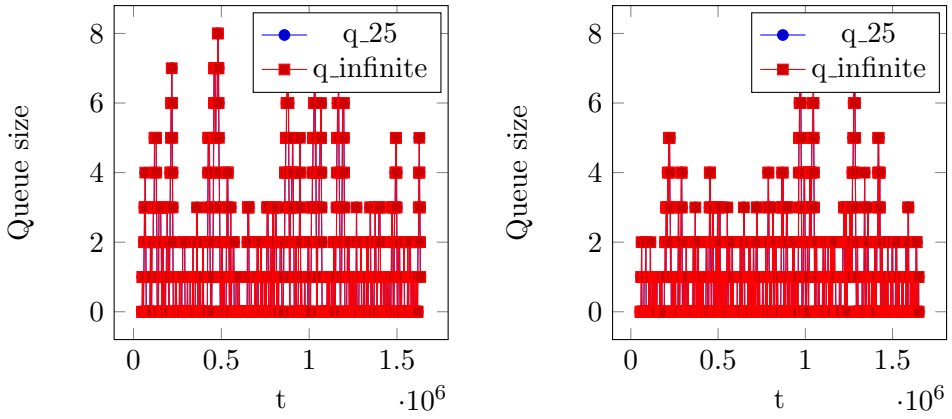
Figure 6.18 Comparing conflicting queue sizes for modified QCSMA



(a) Modified QCSMA Case1 at Node2

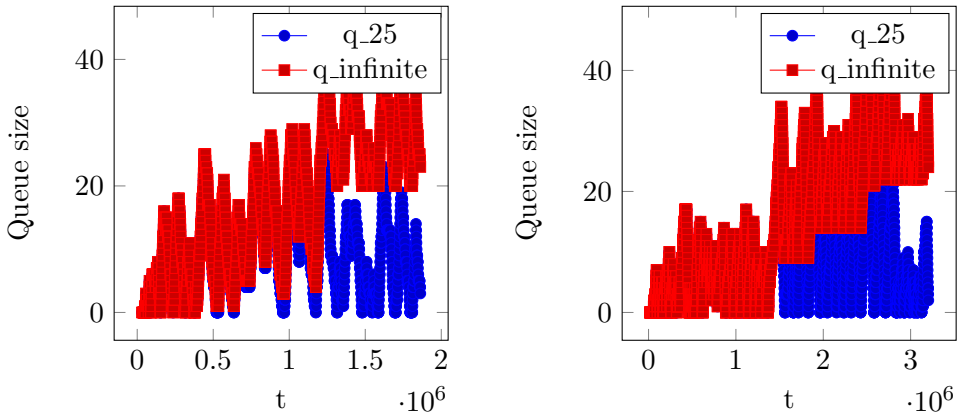
(b) Modified QCSMA Case1 at Node3

Figure 6.19 Queue sizes for Case1 of modified QCSMA



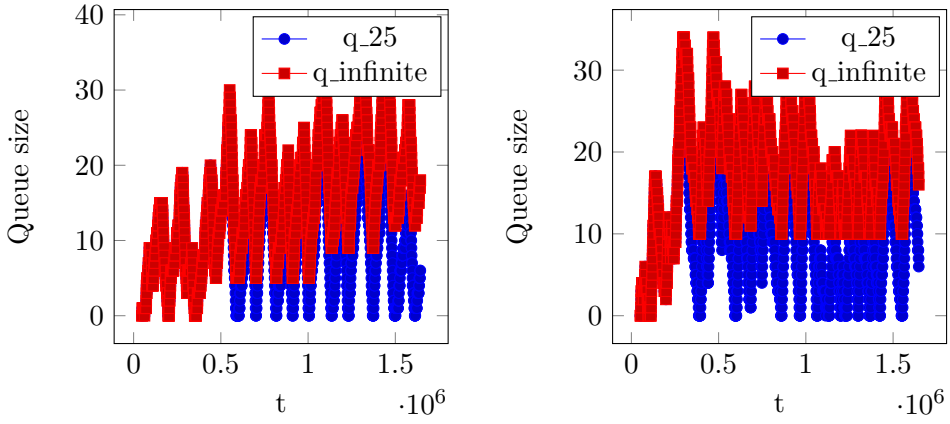
(a) Modified QCSMA Case2 at Node2 (b) Modified QCSMA Case2 at Node3

Figure 6.20 Queue sizes for Case2 of modified QCSMA



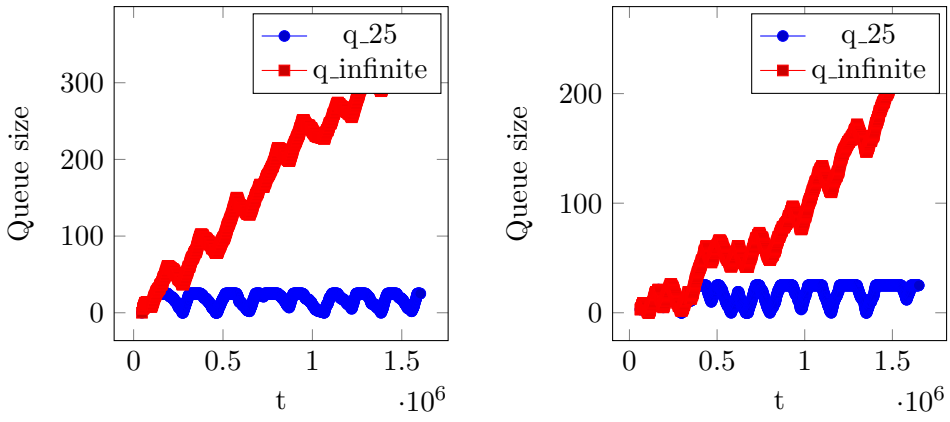
(a) Modified QCSMA Case3 at Node2 (b) Modified QCSMA Case3 at Node3

Figure 6.21 Queue sizes for Case3 of modified QCSMA



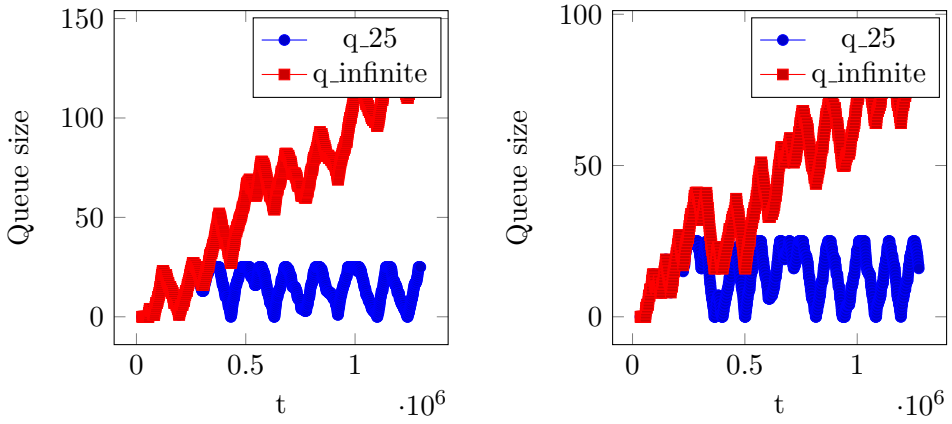
(a) Modified QCSMA Case4 at Node2 (b) Modified QCSMA Case4 at Node3

Figure 6.22 Queue sizes for Case4 of modified QCSMA



(a) Modified QCSMA Case5 at Node2 (b) Modified QCSMA Case5 at Node3

Figure 6.23 Queue sizes for Case5 of modified QCSMA



(a) Modified QCSMA Case6 at Node2 (b) Modified QCSMA Case6 at Node3

Figure 6.24 Queue sizes for Case6 of modified QCSMA

6.3 Conclusions

The packets experienced lesser delay in case of CSMA compared to original QCSMA's line network implementation on telosb motes in Case1 and Case2 when sum of arrival probabilities of two neighboring nodes was much lesser than 1. For cases when the arrival probabilities was almost equal to 1, QCSMA performed better than CSMA for nodes with symmetrical neighbor arrival probabilities. The delay and packets dropped is comparable for the two protocols for Case5 and the packets dropped was 50 percent or lesser in QCSMA compared to CSMA. A compounded data collision effect was observed among the neighboring nodes violating the protocol due to lost packets which was not observed earlier in simulation based protocol comparision. In order to counteract this, a modified implementation was proposed and was tested on the telosb motes in a line network. The peformance of the modified QCSMA was comparable to that of the original QCSMA implementation for some cases and more packets were dropped which could be attributed to the forced turning off of the link whenever a node detected simultaneous data transmissions in the previous time slot. But the data collisions resulting from packet loss was greatly reduced in modified QCSMA compared to QCSMA.

6.4 Future Work

The nodes in the network using the initial synchronization protocol alone may go unsynchronized after large periods of time due to time varying variable such as clock skew or wireless node reset. Resynchronization may be done periodically to prevent such a situation. In this work we used line network to evaluate the performance of the two MAC protocols experiementally. One could extend this work and study the performance of the algorithms in network topologies such as grid, star and ring networks. For this work, the link weights were calculated using a fixed function of queue size. For future work, the link weights as a function of queue sizes can be optimized further for real time networks. Some functions of queue sizes for link weight calculations are suggested in (Ghaderi Srikant) and (Rajagopalan Shah Shin) .

BIBLIOGRAPHY

[Jian Bo Srikant] Jian Ni, Bo (Rambo) Tan, and R. Srikant *Queue-Length Based CSMA/CA Algorithm for Achieving Maximum Throughput and Low Delay in Wireless Networks, Dec. 2011*

[Srikant Lei] R.Srikant, Lei Ying *Communication Networks-An Optimization, Control and Stochastic Networks Perspective, Chapter 5, 2011*

[Tassiulas Ephremides] L. Tassiulas and A. Ephremides *Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks IEEE Trans. Automatic Control, vol. 37, no. 12,pp. 13961948, Dec. 1992*

[Jiang Walrand] L. Jiang and J. Walrand *A distributed CSMA algorithm for throughput and utility maximization in wireless networks In Proceedings 46th Annual Allerton Conference on Communication, Control and Computing September 2008*

[Chaporkar Kar Sarkar] P. Chaporkar, K. Kar, and S. Sarkar *Throughput guarantees through maximal scheduling in wireless networks In Proceedings of 43rd Annual Allerton Conference on Communication, Control and Computing, 2005*

[Boorstyn Kershenbaum Maglaris Sahin] R. R. Boorstyn, A. Kershenbaum, B. Maglaris, and V. Sahin *Throughput analysis in multihop CSMA packet radio networks. IEEE Transactions on Communications, 35(3):267274, March 1987*

[Wang Kar] X. Wang and K. Kar. *Throughput modelling and fairness issues in CSMA/CA based ad-hoc networks. In Proceedings of IEEE INFOCOM, March 2005*

[Ghaderi Srikant] J. Ghaderi and R. Srikant *On the Design of Efficient CSMA Algorithms for Wireless Networks, 2010*

[Rajagopalan Shah Shin] S. Rajagopalan, D. Shah and J. Shin, *Network adiabatic theorem: an efficient randomized protocol for contention resolution, ACM SIGMETRICS/Performance, pp. 133-144, 2009.*

[Crossbow] Crossbow's Telosb DataSheet "www.willow.co.uk/TelosB_Datasheet.pdf" *Document Part Number: 6020-0094-01 Rev B*

[Tinyos] Tinyos Tutorials (2011) "http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Tutorials"

[Tinyos Paper] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, *TinyOS: An Operating System for Sensor Networks, 2004*

[Tinyos Documentation] TinyOS nesc documentation for Telosb platform (2008) "www.tinyos.net/tinyos-2.1.0/doc/nesdoc/telosb"

[Tinyos TEP] Philip Levis (2007), TinyOS 2.x message buffer abstraction "www.tinyos.net/tinyos-2.x/doc/html/tep111.html"